# Structures arborescentes et apprentissage automatique

Marc Tommasi

**Habilitation à diriger des recherches**

soutenue à

l'université Charles de Gaulle — Lille 3

Discipline : Informatique

par

**Marc Tommasi**

---

# Structures arborescentes et apprentissage automatique

---

Date de soutenance : jeudi 23 novembre 2006

Devant le jury composé de :

| | | |
|---|---|---|
| Serge Abiteboul | Directeur de recherches — INRIA Futurs | Rapporteur |
| Patrick Gallinari | Professeur — Université Paris 6 | Rapporteur |
| Marc Sebban | Professeur — Université de Saint-Etienne | Rapporteur |
| François Denis | Professeur — Université de Provence | Examinateur |
| Rémi Gilleron | Professeur — Université de Lille 3 | Examinateur |
| Sophie Tison | Professeure — Université de Lille 1 | Examinatrice |

*à Anne,*
*Marion, Chloé,*
*à Mario, Mireille.*

## Remerciements

Je tiens à remercier Serge Abiteboul, Patrick Gallinari et Marc Sebban d'avoir accepté le rôle de rapporteurs de ce travail et pour l'intérêt qu'ils ont porté à mon travail.

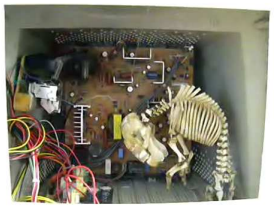
Je remercie Sophie Tison, François Denis et Rémi Gilleron d'avoir accepté d'être présents dans ce jury au titre d'examinateurs.

Sophie Tison m'a engagé sur la voie de la recherche. Elle m'a accordé sa confiance et son soutien, puis m'a formé depuis le DEA et la thèse. J'apprends toujours en discutant avec elle et même lorsque je parviens dans l'instant à décoder ses paroles rapides, ce sont ensuite toujours des semaines d'énergie pour tenter de les comprendre.

Une deuxième rencontre scientifique remarquable et qui m'a beaucoup aidé, est celle de François Denis à l'occasion de mon arrivée à Lille 3. Avoir François comme voisin de bureau et incitateur d'une reconversion thématique en apprentissage automatique est une chance, un catalyseur aussi.

Et dans cet environnement déjà favorisé, j'ai eu la chance d'avoir Rémi Gilleron en supporter permanent. Rémi d'abord en chercheur posé et méthodique m'a formé durant ma thèse. Il m'a apporté la rigueur (le peu de rigueur comparé à la sienne) dans mes premiers pas de chercheur. Il m'a appris à accepter les critiques sur mon travail, critiques souvent si difficiles à recevoir, comme par exemple de retrouver des pages surchargées d'annotations de son stylo rouge. En chercheur curieux, il m'a entraîné dans l'aventure de Grappa et de l'apprentissage automatique. C'est avec beaucoup de plaisir que je constate la facilité avec laquelle nous travaillons ensemble depuis des années, à la fois en enseignement et en recherche. Je le remercie pour son soutien sans faille et toutes ces choses que je n'aurais pu faire sans lui.

Ces remerciements s'adressent bien sûr automatiquement à Alain. J'ai apprécié sa compagnie dans ce bureau de la petite maison. Ses improbables squelettes me rappellent à la réalité, comme ce chercheur qui anime notre bureau, en proie au doute, passant de la rébellion à la résignation face à la machine.


Il règne dans Grappa et Mostrare, une ambiance qui nous permet d'avancer. Cet esprit de groupe, on le doit sans doute à Joachim qui dépense son temps sans compter pour animer notre équipe et nous aider à progresser personnellement.

Je remercie Fabien pour l'aide qu'il m'apporte quotidiennement dans l'organisation du master informatique et document. J'admire son organisation qui lui permet d'écrire des codes d'une lisibilité modèle et même de me rappeler les rendez-vous avant que je ne les oublie.

Je n'oublie pas Isabelle qui n'hésite pas à modéliser la comète de Halley avec un jet de café quand elle décrit le système solaire ni Franck, Daniela, Julien, Patrick, Florent qui ont subi et parfois subissent encore mon interminable apprentissage de l'encadrement de thèse. Je remercie Missi qui a réalisé un travail remarquable dans squirrel et les XCRF.

Enfin, je remercie les membres de l'équipe Grappa pour l'ambiance chaleureuse qu'ils font régner dans la *petite maison*.

# Plan de lecture

Le mémoire comporte cinq chapitres. Dans les quatre premiers j'ai choisi de présenter mes activités de recherche depuis 2003, intégrées dans le projet INRIA Mostrare. Les chaptires de 2 à 4 forment donc un ensemble introduit par le chapitre 1.

Pendant les années qui ont précédé la création du projet Mostrare, j'ai poursuivi une activité de recherche assez soutenue dans les thématiques des langages formels d'arbres et ensuite de l'apprentissage automatique. Ces sujets sont bien-sûr directement liés à la génèse du projet. Cependant, pour affirmer la cohérence du mémoire et sa lisibilité, j'ai choisi de ne les mentionner qu'en dernier chapitre et uniquement par un bref survol.

Les notions d'automates et langages d'arbres ainsi que celles d'apprentissage automatique fondent mes travaux. Pour autant, le mémoire n'en est en aucune façon une introduction ou un exposé précis. Le lecteur pourra consulter par exemple Tata[1] pour les arbres et le livre de Laurent Miclet et Antoine Cornuejols[2] pour l'apprentissage afin d'obtenir à la fois une introduction et mais aussi des informations approfondies sur ces sujets.

[1] `http://www.grappa.univ-lille3.fr/tata`

[2] Apprentissage articiel ; concepts et algorithmes chez Eyrolles

# Table des matières

# Chapitre 1

# Introduction

Ce mémoire présente les dernières années de mon travail de recherche, réalisées au sein de l'équipe Mostrare de l'INRIA Futurs.

Mon travail repose sur deux bases, d'abord les langages formels, plus particulièrement les langages d'arbres, puis l'apprentissage automatique, correspondant chronologiquement à mon parcours scientifique. Je ne peux donc commencer cet exposé sans mentionner un bref historique de mes activités.

J'ai reçu une formation solide grâce à Sophie Tison et Rémi Gilleron lors de ma thèse. Mes goûts des choses plutôt théoriques m'ont certainement incité à choisir les automates d'arbres et les contraintes ensemblistes comme sujet de recherche. La culture acquise pendant cette période me guide encore aujourd'hui. En rejoignant ensuite François Denis et Rémi Gilleron à l'Université de Lille 3, j'ai voulu réaliser une action de mobilité scientifique pour soutenir leur initiative de création d'une nouvelle équipe, Grappa, le Groupe de Recherche en APPrentissage Automatique. Cette double culture apprentissage automatique et automates d'arbres, partagée avec Rémi Gilleron, est sans doute ce qui a pu faire émerger le projet Mostrare quelques années plus tard.

Les travaux présentés dans ce mémoire s'insèrent principalement au sein de Mostrare, à la définition duquel j'ai fortement contribué. Le domaine d'application de Mostrare est l'Internet à travers le traitement automatique de données `XML`, interprétées comme des données arborescentes. Les questions qui m'intéressent sont l'extension ou l'adaptation de méthodes d'apprentissage automatique au traitement de données structurées de façon arborescente. C'est ce que je considère comme le fil rouge et l'originalité de mon travail.

Dans la suite de l'introduction, je vais définir le cadre de travail et les problématiques qui motivent mes travaux actuels. Je résumerai aussi mes contributions avant de les détailler dans les chapitres suivants. Le mémoire se terminera par un bref survol des résultats obtenus en dehors de Mostrare, mais toujours sur les langages d'arbres ou l'apprentissage automatique. Un état de l'art non exhaustif sera à chaque fois reporté en fin de chapitre.

## 1.1 Cadre de travail

À la base de mon travail se trouvent des problématiques d'accès et de manipulation automatiques d'informations au sein d'un réseau d'applications réparties dans l'Internet. Je considère que ces informations sont décrites dans un langage `XML`, et dans la perspec-

tive de ce mémoire, embarquées dans des données structurées sous forme arborescente. Les applications sont basées alors sur des opérations élémentaires que sont l'interrogation ou les requêtes dans ces documents arborescents ou encore la transformation de tels documents.

### 1.1.1 Les données

Bâtir des applications réparties et collaborant au sein d'Internet demande de rendre les données accessibles et interopérables. L'effort de normalisation autour d'XML réalise un pas majeur dans cette direction.

La propriété d'interopérabilité traduit la possibilité de communiquer, de collaborer et d'échanger librement des informations entre plusieurs utilisateurs (humains ou non) quels que soient les environnements matériels et logiciels de chacun. Depuis l'invention et la diffusion de l'informatique, des règles, des normes, se sont établies lentement mais inexorablement pour assurer cette interopérabilité. L'ouverture des normes est aussi un élément déterminant qui permet de les exploiter, les améliorer librement et sans contraintes. L'invention de langages de programmation, de compilateurs, de machines virtuelles, ou encore au sein des réseaux, de normes de protocoles sont autant d'efforts qui participent à la construction de ces règles.

Une tendance aujourd'hui est de porter aussi vers les données et non plus seulement vers les matériels et les protocoles cet effort de standardisation. Le développement d'Internet est indiscutablement lié à l'existence et l'ouverture de ces normes, à partir des protocoles réseaux de base jusqu'aux données comme le langage HTML. Le langage HTML est une représentation de documents qui est exclusivement tournée vers une application particulière d'Internet à savoir la navigation à travers des pages hypertexte. La société a pu constater l'intérêt et l'efficacité de telles normes d'interopérabilité et a rapidement poursuivi cet effort de standardisation des données pour de nombreux autres langages applicatifs. Nous disposons aujourd'hui de langages normalisés pour la représentation d'images vectorielles (SVG), pour les documents de bureautique (ODF) mais aussi pour la musique (MusicXML), les données géographiques (GML), les nouvelles (RSS), les formules mathématiques (MathML),...

Ces langages normalisés ont tous ce même objectif de donner un accès intelligible aux données à la fois aux hommes et aux machines. Issus parfois de SGML, ils reposent maintenant (presque) tous sur la norme XML.

XML est d'abord un effort de normalisation pour la représentation de données numérisées. Pour mon exposé, je ne retiendrai de cette norme que quelques éléments majeurs. Un document XML est un texte balisé. Voici un exemple fictif de document XML qui présente un extrait de la liste des oiseaux du palearctique occidental.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- Oiseaux du palearctique occidental -->
<bdoiseaux>
  <!-- Liste des familles -->
  <familles>
    <famille idfamille="falc">
      Falconidae
    </famille>
```

```
    <famille idfamille="acci">
      Accipitridae
    </famille>
    ...
  </familles>
  <!-- Liste des espèces -->
  <especes>
    <espece idespece="bondapi" reffamille="acci">
      <nom>
        <fr>Bondrée Apivore</fr>
        <en>European Honey-Buzzard</en>
        <lat>Pernis apivorus</lat>
      </nom>
      <photo>bondree.jpg</photo>
      <description>
        De la taille d'une <oiseau refespece="busevar">buse
        variable</oiseau>, la bondrée cercle à plat au dessus des
        bois.
      </description>
    </espece>
  </especes>
  ...
</bdoiseaux>
```

Les données XML ont plusieurs caractéristiques intéressantes qui participent aux objectifs d'interopérabilité, d'ouverture et d'automatisation d'Internet.

- Elles sont intelligibles : À l'image d'un code source de programme, la lecture d'un document par un humain est compréhensible et peut faire sens. Pour les programmeurs et les responsables de contenus, c'est une caractéristique importante puisqu'elle permet d'accéder facilement aux informations portées dans le document et d'écrire plus simplement différents programmes les exploitant.
- Elles s'autodécrivent : les balises portent (souvent) par leur nom une information qui peut être interprétée comme une méta-information sur les données qu'elles encadrent. Pour aider cette interprétation sémantique à la fois par les humains ou les programmes, le recours à des ontologies (elles mêmes décrites en XML...) et des raisonnements sur ces interprétations peuvent même être envisagés.
- Elles sont extensibles mais peuvent être typées. Le typage apporte la sûreté pour l'exécution des programmes puisqu'il permet, dans une certaine limite, d'éviter les erreurs dues à une donnée de valeur imprévue. Dans le cadre d'XML, le typage est suffisamment souple pour ne pas pénaliser la variété des valeurs que l'on souhaite représenter.

Il en résulte plusieurs conséquences importantes pour mon travail. Le caractère intelligible des données XML et leur faculté de s'autodécrire permet d'envisager de réaliser plus de traitements purement automatiques. Une illustration peut être donnée par le fameux exemple de Berners-Lee et al. (2001). Je détaille cet exemple ci-dessous pour introduire les tâches considérées dans ce mémoire. Par contre, le caractère extensible des données

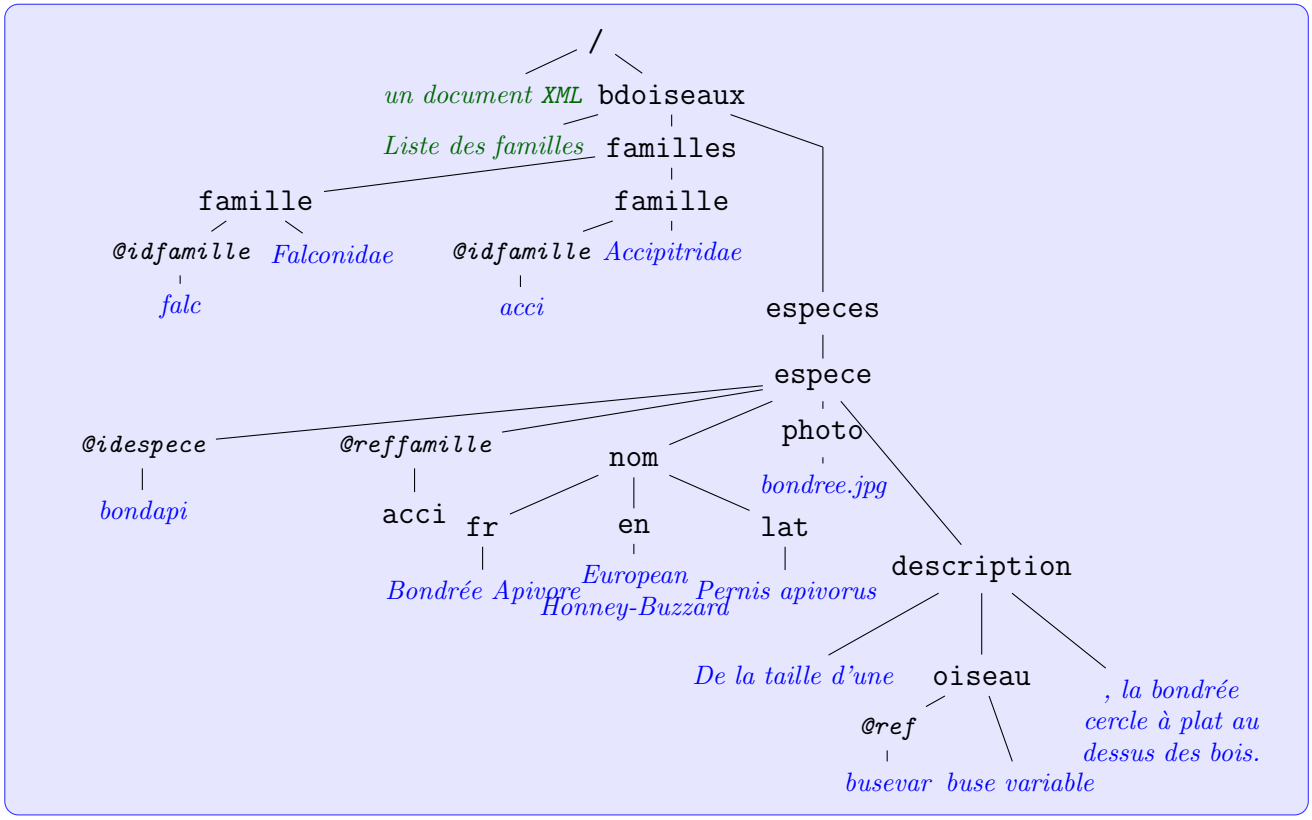



FIG. 1.1 – Vue arborescente du document XML page 2.

entraîne des difficultés dans la réalisation de ces tâches automatiques. L'extensibilité va se traduire pour nous en un problème d'hétérogénéité des données, renforcée par leur nature distribuée, et par le fait qu'elles sont générées par des programmes ou auteurs différents.

Une autre caractéristique d'importance pour mon travail est que l'organisation des balises est telle qu'on peut représenter le document comme un arbre (l'arbre associé au document de la page 2 est dessiné dans la figure 1.1). C'est un premier choix. On pourrait aussi le voir comme une séquence : séquence de bits, de caractères ou d'unités lexicales ; comme un graphe à cause de la sémantique implicitement donnée par les mots `idfamille` et `reffamille`.

Cette vue arborescente est standardisée dans la recommandation DOM du W3C. La recommandation XML distingue différents types de noeuds dont certains se retrouvent dans l'exemple. On peut retrouver : les noeuds **élément** correspondant à une balise, les noeuds **attributs** désignant un attribut et sa valeur, les noeuds **commentaire**, les noeuds **texte** contenant des données textuelles. Notez aussi que certains types de noeuds ont un **contenu mixte** mêlant à la fois texte et éléments. Ce sont essentiellement les noeuds élément, attribut et texte qui sont porteurs de données. Dans la suite nous oublierons le noeud `/` (en racine) et les noeuds porteurs de commentaires.

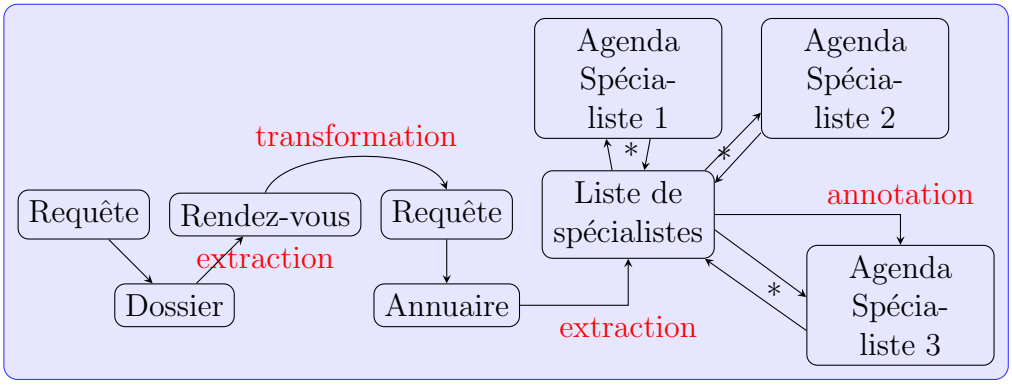



FIG. 1.2 – Les documents échangés entre l'agent de Lucie et les services Internet des autres acteurs. Les échanges de documents `XML` sont enchaînés avec des extractions, transformations ou annotations de documents. (Les parties avec * ne sont pas détaillées).

### 1.1.2 Les tâches

Dans Berners-Lee et al. (2001), les auteurs envisagent un nouvel Internet et tracent les contours du web sémantique. Ils illustrent leur propos à l'aide d'un exemple que je reprends et adapte ici. La maman de Lucie doit voir un spécialiste et suivre un ensemble d'examens. Lucie utilise, à partir de son assistant personnel, un agent sur Internet qui réalisera l'ensemble des opérations. L'agent sera en charge de récupérer le dossier de sa mère, trouver un spécialiste libre dans la semaine, proche géographiquement et compatible avec son assurance et programmer les examens.

Pour que cet agent puisse fonctionner, il faut que les données dont il a besoin soient accessibles. C'est par exemple le cas des carnets de rendez-vous des spécialistes. On imagine alors que les données sont représentées en `XML`. Il faut aussi que l'agent soit capable d'effectuer des opérations comme interroger un annuaire de spécialistes, y localiser les caractéristiques de chacun, interroger les disponibilités d'un spécialiste, demander d'y écrire une réservation.

L'article de Berners-Lee et al. se poursuit par un argumentaire sur les méta-données facilitant l'interprétation des données par les programmes : les données au format `RDF`, les ontologies et les règles d'inférences attachées à toutes ces informations sémantiques. Cette partie est hors du sujet de mes activités de recherche. Aujourd'hui ces informations sémantiques sont encore trop peu souvent disponibles. Le raisonnement à l'aide de ces informations peut être mis en difficulté par le passage à l'échelle et la complexité intrinsèque de ce calcul. Par contre, la réalisation de tâches complexes nécessitant l'interopérabilité et la coopération d'agents est aujourd'hui répandue. On peut citer par exemple les sites de comparaison de prix ou de services sur Internet qui présentent sur un portail unique une information homogénéisée, obtenue à partir de plusieurs sources.

Comme l'illustre la figure 1.2, bâtir de telles applications demande de construire des requêtes sur des données `XML` et d'extraire des informations, mais aussi de les annoter ou les transformer. Par exemple, l'agent de Lucie commence par interroger le dossier de sa maman. Une extraction d'un rendez-vous de ce document que l'on suppose stocké en `XML` est ensuite effectuée. Pour obtenir la liste de spécialistes, le fragment de `XML` qui correspond au rendez-vous est transformé de façon à composer une requête, intégrant les contraintes d'assurance par exemple. Soumise à un service d'annuaire, la requête

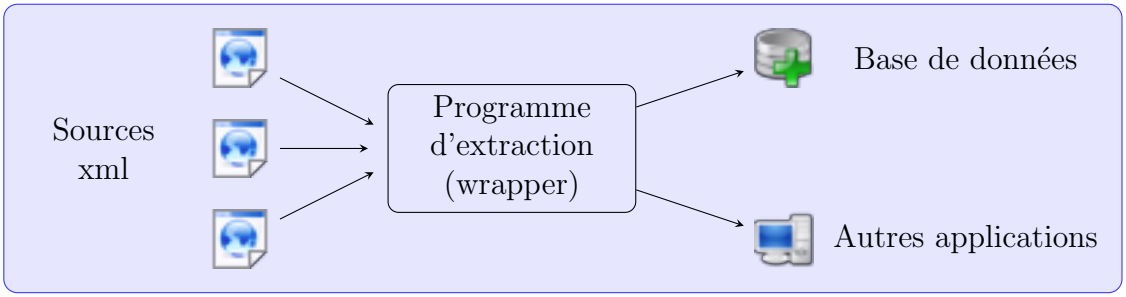



FIG. 1.3 – Extraction. Des documents en entrée, une base de données en sortie d'un programme d'extraction appelé wrapper.

donne lieu à une réponse qu'il s'agit d'analyser pour en extraire la liste de spécialistes auxquels il sera possible de s'adresser. L'extraction peut par exemple aussi prendre en compte le critère de localité. Ainsi de suite, le transport d'informations au format `XML` de services en services sera réalisé par une succession de tâches de base comme l'extraction, l'annotation ou la transformation. Cette liste de tâches n'est pas exhaustive mais ce sont à ces opérations que je vais particulièrement m'intéresser.

### Extraction

La problématique de l'extraction dans les documents `XML` est celle d'isoler des parties de documents pour, souvent, alimenter une base de données. La présence d'une base de données permet ensuite d'accéder aux informations tout en restant hors-ligne et de façon efficace. La tâche considérée n'est pas véritablement celle de l'extraction, mais plutôt celle de la réalisation de programmes d'extraction (appelés wrappers dans la communauté). C'est une tâche étudiée depuis longtemps par des spécialistes de linguistique informatique et du traitement automatique des langues. Le besoin est apparu pour l'exploitation de données textuelles écrites en langage naturel dans de grands corpus. La problématique est toujours d'actualité mais se décline aussi aujourd'hui en des tâches un peu différentes quand il s'agit de données `XML`. Essentiellement, la tâche se rapproche de celle de la construction de requêtes et reste très syntaxique alors qu'elle demande une analyse profonde du sens dans le cas des textes en langage naturel.

Dans l'univers d'`XML`, la définition précise et la difficulté de cette tâche sont très dépendantes de nombreux paramètres. J'en retiens quelques uns qui seront discutés dans la suite de ce mémoire.

- Les données peuvent être plus ou moins homogènes. Les sites Internet modernes que nous considérons comme sources de nos données, sont maintenant souvent générés par programme. Il en résulte une grande homogénéité des documents provenant de la même source. Lorsque plusieurs sources sont utilisées pour alimenter une base de données, plusieurs outils d'extraction sont souvent constitués. Par contre, l'évolution des sources, dues à l'évolution des technologies, ou un changement de charte graphique par exemple pose le problème de la maintenance des programmes d'extraction.
- Les données peuvent être plus ou moins structurées. Le format `XML` permet une grande variété de structuration : depuis les documents de type textuels où toute l'information se trouve dans des parties de texte libre très grandes jusqu'aux docu-

ments de type bases de données où les informations sont très découpées et isolées dans des feuilles distinctes de l'arbre XML.
- Le type de l'information à extraire joue aussi un rôle important. Le type le plus simple à traiter est une donnée atomique : on cherche à extraire plusieurs occurrences d'une seule donnée comme l'heure de disponibilité du spécialiste dans le cas de Lucie. On parle alors d'**extraction monadique**. Quand il s'agit d'extraire des données en relation, on parle alors d'**extraction $n$-aire**. C'est par exemple l'extraction des occurrences de couples donnant la date et l'heure. Nous verrons dans le chapitre 3 que le passage de l'extraction monadique à l'extraction $n$-aire est une réelle difficulté. Même disposant d'extractions monadiques parfaites des composantes d'une relation, la réassociation des composantes pour former des $n$-uplets complets de la relation pose des questions non triviales.
- Certaines applications comme l'agent de Lucie, peuvent ne pas connaître exactement le format du carnet d'adresse de certains spécialistes. Le type exact des données à extraire qui sont requises pour, par exemple, interroger ces carnets d'adresses peut donc ne pas être considéré comme une entrée de la procédure d'élaboration des outils d'extraction.

### Transformation

L'agent de Lucie doit enchaîner des tâches d'interrogation et d'analyse des résultats, résultats qui seront aussitôt à la base d'une interrogation suivante. Dans ce cadre, on peut dire que l'agent réalise une transformation qui est ici une transformation de documents XML. Cette interprétation du traitement est encore plus évidente dans le cas de processus d'intégration. L'intégration consiste à relever des données provenant de sources distinctes et de les insérer dans un système d'information unique. C'est un processus d'homogénéisation qui s'accompagne souvent d'un gain de sens et donc une valorisation des données. Dans l'exemple de la figure 1.2, l'annuaire peut conserver les résultats d'extraction formulés auprès des agendas de spécialistes dans une base de données unique, accélérant ainsi les temps de réponses de futures requêtes. La transformation de documents aux formats variés (tels que PDF ou HTML) vers un format unique XML est aussi une illustration du processus d'intégration.

### Annotation

Réaliser des programmes de transformation directe de XML en XML ou d'extraction n'est pas un exercice facile. Nous isolerons dans ces processus une brique élémentaire qui consiste à annoter un document XML. La tâche d'annotation consiste à décorer un document avec des étiquettes. Les étiquettes peuvent avoir une sémantique particulière qui permet d'envisager des tâches plus complexes. Par exemple, des tâches d'extraction monadique peuvent être réalisées avec des annotations booléennes signifiant « à extraire » ou « à ne pas extraire ». Pareillement, des transformations sont aussi décrites par des annotations qui peuvent signifier « dupliquer cette partie », « supprimer », « inverser »... Une illustration sous forme d'exemple de ces annotations est donnée dans quelques pages en section 1.2.4.

**Constat**

Aujourd'hui, les manipulations d'XML comme la transformation ou l'extraction sont réalisées à l'aide de technologies reposant sur des langages standardisés comme XPath, XQuery, XSLT. La dimension arborescente des données XML est clairement exploitée dans ces langages. Pour les avoir enseignés à des publics non informaticiens ou simplement débutants, je peux témoigner qu'une solide culture algorithmique, notamment sur les arbres, ainsi qu'une capacité à manipuler des langages formels sont nécessaires. La réalisation de tels agents est donc réservée à des publics d'informaticiens confirmés.

Cependant deux raisons m'amènent à penser qu'il est nécessaire de simplifier la réalisation de tels outils d'extraction, transformation ou d'annotation. La première est la rapidité d'évolution, d'apparition et de disparition des sources sur Internet. Ceci entraîne une réécriture continue des programmes, qui doit se faire dans un temps le plus court possible pour limiter l'interruption des services ou pour garder un caractère actuel dans les résultats finaux. La seconde est la volonté de donner à tout utilisateur non informaticien la possibilité de construire ces programmes. Ceci inclut aussi leur construction automatique pour assurer un service d'agents autonomes.

Dans l'ensemble de ces tâches, des événements imprévus ou incertains se produisent. Par exemple, l'évolution d'une source de données, l'apparition de cas particuliers peuvent altérer la fiabilité des systèmes construits ; certaines manipulations exploitent des données textuelles et sont sujets à toutes les imprécisions dues aux outils de traitement de la langue naturelle. On ne peut pour autant se passer de ces informations.

L'agent de Lucie doit recevoir et peut donner des résultats faux à chaque étape de son processus de recherche. Il ne peut les écarter mais doit s'adapter. Le processus d'intégration doit aussi tenir compte de plusieurs alternatives issues de bruit ou de procédures non déterministes.

Tenir compte d'incertitudes est la dernière dimension dans la difficulté des tâches que je souhaiterais aborder. Elle donne aussi un élément de lecture du travail que j'ai réalisé, commençant par des méthodes exactes et se terminant par des outils probabilistes.

## 1.2 Contributions

Mes activités de recherche ces dernières années s'inscrivent dans le projet INRIA Mostrare et aussi dans le cadre d'une ACI (Action Concertée Incitative) Masse de données intitulée Marmota. Les travaux ont été réalisés en équipe pendant les thèses en cours ou passées de Julien Carme, de Patrick Marty et de Florent Jousse. Je décris maintenant les contours de ces projets et les résultats obtenus.

### 1.2.1 Mostrare et Marmota

Notre proposition au sein du projet Mostrare a été de recourir à des techniques d'apprentissage automatique pour faciliter la réalisation de tâches d'extraction, de transformation de données XML. Le projet s'est développé selon deux axes : l'un poursuivant les recherches sur les langages formels d'arbres, et leur adaptation en regard des spécificités de XML ; l'autre étudiant la question de l'apprentissage lorsque les données sont structurées de façon arborescente. Les deux axes poursuivent aussi ce but commun de réaliser

les tâches décrites ci-dessus.

L'idée du projet Mostrare s'est construite pour moi à partir de la réflexion initiée par le sujet de thèse de Julien Carme. Rémi Gilleron et moi même avions à ce moment la volonté d'étudier les questions de l'apprentissage automatique en présence de données arborescentes. Nous étions persuadés de l'intérêt de ces travaux pour la manipulation de données de l'Internet. L'idée a pris la dimension de projet quand avec Sophie Tison nous l'avons étendue à l'étude des langages d'arbres pour l'Internet et dans le but de les apprendre automatiquement. L'arrivée ensuite de Joachim Niehren a donné un objectif plus précis au projet à savoir l'extraction d'informations à partir de données arborescentes.

Dans un premier temps des approches purement syntaxiques ont été développées. La thèse de Julien Carme a abouti à une application d'extraction d'information interactive pour des problèmes monadiques à l'aide d'inférence de langages d'arbres. Puis dans la thèse de Patrick Marty, nous avons proposé une relaxation de ces méthodes exactes en se reposant sur des techniques bien étudiées de classification supervisée. Enfin, dans la thèse de Florent Jousse nous étudions les modèles probabilistes de langages d'arbres. Ces trois parties ne regroupent pas la totalité du projet Mostrare mais uniquement les actions dans lesquelles je me suis particulièrement impliqué.

Par ailleurs, le travail de Florent s'inscrit dans une action ACI intitulée Marmota [1] dont je suis responsable scientifique. L'opportunité de ce projet a été de redéfinir certains objectifs de Mostrare dans le champ des langages probabilistes et des statistiques inférentielles. Marmota regroupe quatre équipes de recherche : l'équipe de Patrick Gallinari au LIP6, l'équipe de Marc Sebban à Saint-Etienne, l'équipe de François Denis à Marseille et le projet Mostrare. Le sujet de ce projet se situe à l'intersection de trois domaines de recherche : les langages formels d'arbres, l'apprentissage automatique, les modèles probabilistes. Les applications principales de ces recherches sont liées aux technologies `XML`, notamment : l'intégration de données du Web depuis des sources hétérogènes et distribuées, l'annotation et la transformation de données `XML`, la classification et la segmentation de documents `XML`. Cependant, les avancées attendues auront un impact important dans les domaines où les notions de structure arborescente et de probabilités ont une grande importance. C'est le cas en bioinformatique et en indexation musicale, où la représentation arborescente des données est très pertinente. Les recherches concernent plus spécifiquement :

- les modèles génératifs pour les données arborescentes : grammaires probabilistes pour `XML`, algorithmes d'analyse syntaxique et induction de telles grammaires ;
- les modèles non génératifs pour les données arborescentes : champs aléatoires conditionnels pour les arbres, algorithmes d'annotation, modèles discriminatifs et noyaux d'arbres, algorithmes de classification ;
- les transformations probabilistes d'arbres et l'alignement probabiliste de motifs d'arbres probabilistes, l'apprentissage automatique de tels motifs et transformations.

J'introduis dans les trois sections suivantes les travaux menés dans les thèses de Julien Carme, Patrick Marty et Florent Jousse. Ces sujets seront approfondis dans les chapitres

[1] MAchine learning pRobabilistic MOdels Tree lAnguages. C'est aussi le nom latin d'une marmote qui escalade les arbres et ressemble alors dans son comportement à un gros écureuil, (squirrel en anglais et acronyme d'un logiciel d'extraction de notre groupe.

suivants.

### 1.2.2 Automates d'arbres et extraction d'information

Le sujet de Julien abordait à la fois l'étude de langages d'arbres adaptés à la représentation de données XML et leur apprentissage exact par des techniques d'inférence grammaticale. Julien a pu ensuite appliquer les résultats de sa thèse dans un prototype pour l'extraction d'information monadique.

Dans la littérature sur les langages d'arbres, de nombreux travaux ont été consacrés aux arbres d'arité bornée. Les noeuds de ces arbres sont décorés avec des symboles qui déterminent de façon fixe le nombre de fils sous chaque noeud. Ces arbres d'arité fixe sont souvent appelés des **termes**. Dans cette classe de langages de termes beaucoup de résultats théoriques ont été établis. Notamment, c'est dans cette classe qu'on définit des automates d'état finis pour les représenter et les manipuler. Les arbres XML ne sont pas des termes. Chaque noeud a un nombre de fils qui n'est pas borné a priori. Un premier travail a été de déterminer une formalisation et un cadre théorique pour manipuler les arbres d'arité non bornée. Plusieurs formalismes existent déjà, mais nous semblaient peu adaptés en regard des objectifs d'apprentissage automatique que nous nous étions fixés. Nous avons proposé une réduction que je qualifie de simple et éclairante du cas de l'arité non bornée au cas classique des termes. Cette réduction tout à fait algébrique a facilité la poursuite de nos travaux.

En obtenant cette réduction au cas des termes, nous bénéficions gratuitement de nombreux résultats fondamentaux sur la déterminisation, la minimalisation de tels langages. Du point de vue de l'apprentissage automatique, nous obtenons aussi directement des algorithmes d'inférence de langages d'arbres d'arité non bornée à partir de ceux traitant de termes.

L'application à l'extraction d'information est réalisée en considérant alors des langages de termes dont les symboles comprennent une information booléenne « à extraire » ou « à ne pas extraire ». L'apprentissage d'un outil d'extraction est alors réduite à un problème d'inférence grammaticale : apprendre un langage d'arbres représenté sous la forme d'un automate ou d'une grammaire. Les programmes d'extraction sont représentés par des automates d'arbres qui désignent des ensembles de noeuds.

Le prototype développé lors de la thèse de Julien est intégré sous forme d'extension au navigateur Firefox. Plus tard, le développement a été repris pour réaliser une plate-forme client serveur. Un serveur applique les programmes d'apprentissage. Le client actuel est toujours une extension de Firefox. D'autres clients sont en développement sous forme de service Web.

### 1.2.3 Classification supervisée et extraction d'information

Considérer des documents XML comme source de données, c'est aussi envisager une structure, l'arbre DOM, plus complexe qu'un arbre d'arité non bornée. Les données XML contiennent des attributs, et d'autres types de noeuds pouvant contenir des données importantes pour nos tâches d'extraction ou de transformation. Les attributs sont des noeuds qui ne sont pas ordonnés entre eux. D'autre part, dans le cas d'une tâche d'extraction, les données à extraire sont parfois localisées dans des feuilles ou des parties de

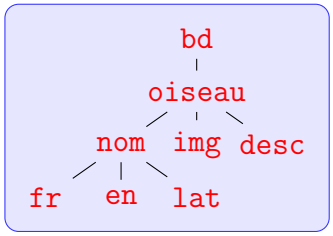



FIG. 1.4 – Exemple de modèle de donnée pour le résultat d'une transformation à partir de plusieurs sources de données.

feuilles de cet arbre DOM et pas seulement dans des noeuds. Enfin, la variété des sites web font que certaines irrégularités peuvent se produire sur la structure des pages.

Toutes ces considérations mettent en difficulté l'approche par inférence grammaticale développée par Julien. Une alternative a été pour nous de considérer l'utilisation de méthodes ne reposant pas exclusivement sur la structure arborescente des données.

L'idée principale dans la thèse de Patrick Marty est d'utiliser une représentation des données exploitant à la fois leur dimension arborescente mais aussi la dimension textuelle obtenue par la concaténation des feuilles de l'arbre. Cette représentation est compilée dans un ensemble d'attributs. Un arbre XML est alors représenté comme un ensemble d'enregistrements, comme autant de constituants qu'il s'agit d'identifier comme « à extraire » ou « ne pas extraire ».

Grâce à cette représentation, des algorithmes de classification, bien étudiés dans la littérature, peuvent être utilisés pour déterminer les parties à extraire. Nous avons étudié l'impact de la qualité de la représentation sur la tâche d'extraction pour nous aider à déterminer le jeu d'attributs optimal à la fois sur des données textuelles ou arborescentes. Nous avons validé l'usage d'algorithmes de classification sur des problèmes d'extraction monadique. Nous avons ensuite poursuivi dans cette voie en proposant un algorithme pour l'extraction de données issues d'une relation $n$-aire.

Pour le problème d'extraction $n$-aire, nous avons étudié les différentes façons de stocker des occurrences (tuples) d'une relation dans un arbre. C'est une opération souvent réalisée par les générateurs de rapports ou tableaux de bord de suites décisionnelles ou de gestionnaires de bases de données qui produisent du XML ou HTML. Cette étude nous a amené à proposer un algorithme original d'extraction, incrémental et interactif.

L'algorithme a été implanté et intégré dans la plate-forme d'extraction.

### 1.2.4 Modèles probabilistes et annotation d'arbres

Dans le travail de Julien, le problème d'extraction dans les arbres a été traduit dans celui de l'identification d'un langage d'arbres étiquetés par une information booléenne. Clairement, il est facile de ne plus se limiter à un choix d'étiquettes dans un ensemble à deux valeurs. C'est l'approche la plus immédiate pour aborder le problème de la transformation d'arbres. Supposons comme dans la figure 1.3, que nous devions mémoriser des informations provenant de différentes sources dans un modèle unique. Si ce modèle est arborescent comme par exemple celui illustré dans la figure 1.4, alors l'intégration de ces données s'effectue par une transformation d'arbres.

Pour apprendre à partir d'exemples la transformation à réaliser pour chaque source, on peut tenter d'apprendre le langage des superpositions arbre origine, arbre transformé

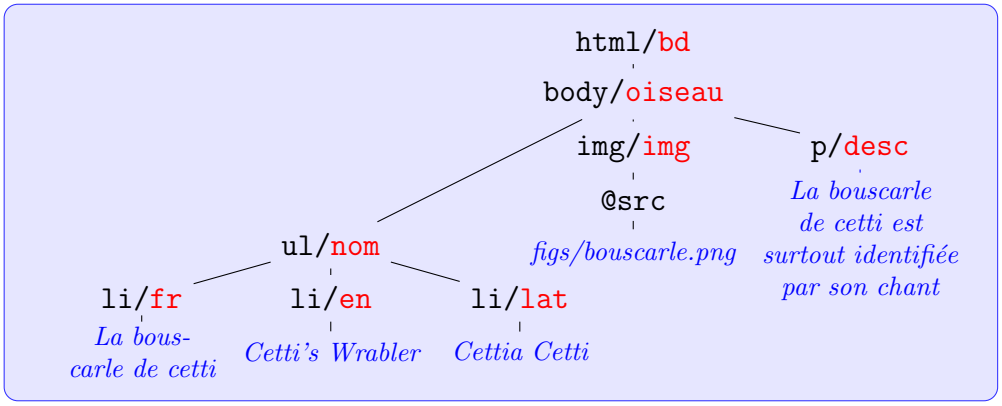



FIG. 1.5 – Superposition d'arbres. L'arbre d'entrée et l'arbre transformé sont de structures proches. On forme un nouveau langage d'arbres sur l'alphabet des couples qu'on peut apprendre.

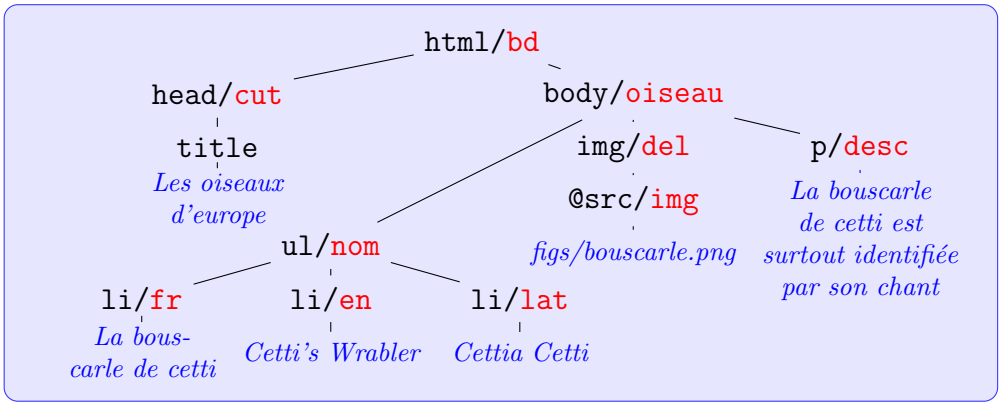



FIG. 1.6 – Annotations d'édition. On introduit des étiquettes associés à une sémantique particulière. `Del` et `Cut` coupent des parties d'arbres.

comme indiqué dans la figure 1.5. La forme de l'arbre d'entrée doit rester très proche de celle de l'arbre de sortie.

On peut aussi représenter la transformation par un ensemble d'opérations d'édition réalisées à chaque noeud et apprendre le langage des arbres étiquetés par ces opérations (figure 1.6).

Du point de vue de la théorie des langages d'arbres et de l'inférence grammaticale exacte, ces deux idées seront certainement étudiées dans l'équipe. Mais l'expressivité de la classe des transformations apprenables sera sans doute limitée. Compte tenu de la potentielle diversité des sources une annotation même incomplète comme celle illustrée dans la figure 1.7 sera nécessaire. En tout état de cause, on traduit de cette façon un problème de transformation en un problème d'annotation. Il nous est donc apparu essentiel de poursuivre dans l'apprentissage automatique de programmes d'annotation.

Nous étudions le problème d'inférence de modèles stochastiques d'annotations d'arbres dans la thèse de Florent.

Dans le cas de séquences, le problème d'annotation a été largement étudié. Les modèles les plus utilisés sont certainement des chaînes de markov cachées (`HMMs`). Les `HMMs` sont des modèles probabilistes génératifs assez proches d'automates probabilistes. Ils

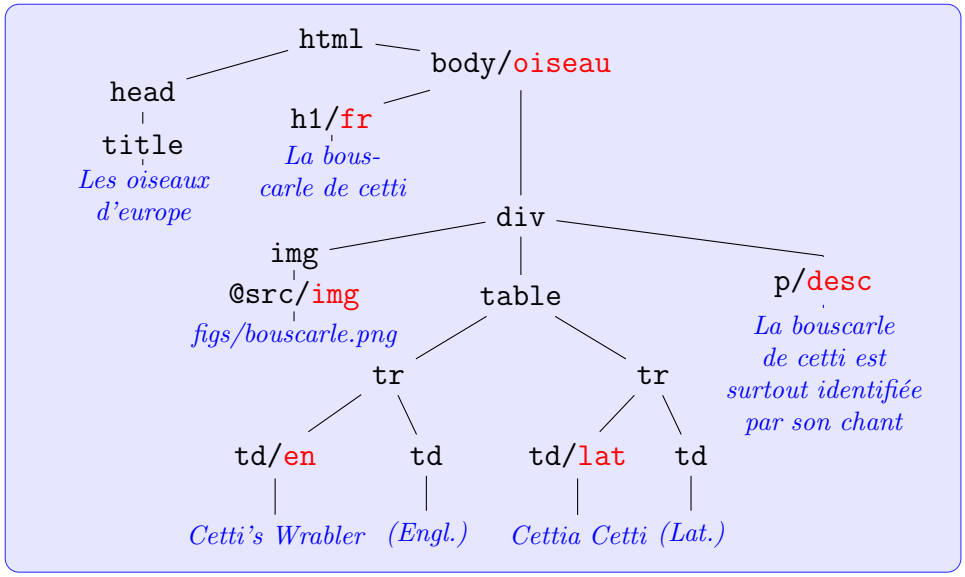



FIG. 1.7 – Exemple d'annotation. L'annotation est partielle mais peut fonctionner sur toute forme d'arbre en entrée.

représentent la distribution de probabilité qui régit la génération des données. Pour la tâche d'annotation, la séquence d'états (cachés) par lesquels le HMMs transite pour générer la séquence observée sera la séquence d'annotations. La modélisation de la probabilité de génération des observations est une opération qui peut être difficile et qui n'est pas nécessaire ici. Les approches non génératives, contournent cette difficulté et me semblent plus adaptées dans le cas d'une tâche d'annotation.

Les champs conditionnels aléatoires (CRFs) représentent des distributions conditionnelles. Ce sont des modèles non génératifs qui ont été appliqués avec succès sur des séquences pour des tâches d'extraction. Un CRF représente alors la distribution conditionnelle d'obtenir une annotation connaissant l'observation.

Nous avons adapté le modèle des CRFs au cas des arbres XML. Nous avons obtenu un modèle permettant d'annoter les différents types de noeuds d'un arbre XML (éléments, textes ou attributs) ordonnés ou pas.

Le travail a donné lieu à un développement maintenant disponible sur la forge de l'INRIA [2]. Appliqué à des tâches d'extraction ou de simples transformations, notre approche donne des résultats très encourageants, dépassant l'état de l'art.

## 1.3 Notes bibliographiques

### 1.3.1 Les données XML

Le W3C, les instituts de normalisation comme l'OASIS ainsi que de nombreux groupements d'intérêt contribuent fortement à la naissance, la maintenance et la diffusion de standards reposant sur XML. Ces standards sont souvent le prologement de normes conceptuelles et parfois d'implantations réalisées en SGML dans les années 80 ou 90. Internet a dynamisé et renforcé cette tendance, insistant particulièrement sur la nature

[2] treecrf.gforge.inira.fr

distribuée des sources. Les définitions des langages sont publiques et disponibles sur les sites Internet de ces instituts[3]. Une approche plus théorique des données de l'Internet est rassemblée dans le livre d'Abiteboul et al. (2000).

L'évolution dans le temps des formes de typage des données `XML` à travers les `DTD` puis les schémas et `RelaxNg` illustrent aussi la transformation d'Internet vers plus d'automatisation. D'un point de vue théorique, les `DTD` sont fortement inspirées des grammaires de mots et des travaux de type sur `SGML`, sont peu expressives et leur développement certainement guidé par la volonté d'utiliser un certain type d'algorithmes d'analyse. Les langages sont 1-non ambigus (voir Bruggemann-Klein and Wood, 1998). Les schémas s'adaptent mieux à des développements d'applications plus sûres raffinant le typage et en intégrant par exemple les espaces de noms plus sérieusement que dans les `DTD` (Clark (1999); Bourret (2005)). Mais leur justification théorique est peu claire (Bex et al., 2004). Un beau compromis pour moi est l'approche centrée sur les automates d'arbres de `RelaxNg` réalisée par l'unification du `TREX` (Tree Regular Expressions for `XML`) par J. Clark et de `Relax` (REgular LAnguage description for `XML`) par M. Makoto.

Les langages de transformation et d'annotation ou de requête dans les arbres `XML` sont des standards du `W3C` (`XSLT`, `XPath`, `XQuery`). Ils sont aujourd'hui intégrés dans des processus industriels et des produits grand public. Mais ces langages sont parfois critiqués pour des raisons théoriques aussi bien que pratiques. Par exemple, des tâches simples ne se traduisent pas toujours facilement. Ré-étiqueter un arbre `XML` en `XSLT` est laborieux. D'autre part la réalisation de grandes applications reposant sur `XSLT` ou `XQuery` se heurte rapidement à quelques principes de génie logiciel et la maintenance peut devenir difficile. Des alternatives sont toujours proposées aujourd'hui pour tenter de pallier ces difficultés. Citons Xduce, TOM et Transformer, trois projets au sein de l'INRIA. La plupart des applications `XML` reposent sur `XPath` qui permet essentiellement de désigner des ensembles de noeuds dans un document. Ce langage pivot a lui même fait l'objet de nombreuses études bien après sa définition. Les travaux de Gottlob et al. (2003) donnent quelques années après la publication de `XPath` 1.0 une mesure précise de la complexité combinée des requêtes. Les mêmes auteurs ont ensuite proposé des algorithmes efficaces en 2005 alors que de nombreuses implantations très populaires restent exponentielles.

### 1.3.2 Les tâches

Tous ces langages de transformation, mais aussi les langages de programmation classiques intégrant la manipulation d'expressions régulières de mots (comme `Perl` ou `Python`) sont massivement utilisés pour réaliser les actions décrites dans ce chapitre. L'emploi de techniques d'apprentissage automatique est apparu aussi assez rapidement, exploitant des travaux plus anciens mais toujours actifs sur les données purement textuelles.

C'est particulièrement le cas pour l'extraction d'information. Fortement lié au domaine du traitement automatique de la langue naturelle, et aussi dans la communauté des bibliothèques numériques, l'extraction d'information repose alors souvent sur des définitions d'expressions ou de motifs sur les séquences. Le livre reposant sur le travail de thèse de Poibeau (2003) en donne une bonne illustration. Dans ces travaux les connaissances linguisitiques et du domaine des textes sont directement insérées dans le système

[3] `www.w3.org` et `www.oasis-open.org`

d'extraction par l'utilisateur sous la forme de définitions de motifs à l'aide d'automates ou de lexiques, dictionnaires, ... L'apprentissage automatique a été envisagé pour la construction automatique de ces motifs dans des systèmes comme rapier de Califf (1998) ou whisk de Soderland (1999) ou encore SRV de Golgher et al. (2001).

Mais dans le domaine d'Internet ce sont certainement les travaux de N. Kushmerick (Kushmerick, 1997, 2000; Freitag and Kushmerick, 2000) qui ont lancé la dynamique de recherche mêlant apprentissage automatique, Internet et extraction d'information. On peut citer par exemple W4F de Sahuguet and Azavant (2001), RoadRunner de Crescenzi et al. (2001), XWRAP de Liu et al. (2000) ou l'article de synthèse de Laender et al. (2002). De façon surprenante, ces travaux n'exploitent pas ou peu la structure arborescente des données mais seulement la présence de balises. Par contre, Stalker par Muslea et al. dès 1998 puis WL2 de Cohen et al. (2002) exploitent l'information arborescente des documents, mais sans avoir recours au support des automates d'arbres à la différence de Kosala et al. (2002).

Parallèlement, les travaux de l'équipe de G. Gottlob (Gottlob and Koch, 2004, 2002) ont particulièrement animé et modernisé la compréhension de la tâche d'extraction depuis des données XML. Leurs idées sont rassemblées et implantées dans le logiciel Lixto (Baumgartner et al., 2001; Gottlob et al., 2004). Ce système n'utilise pas de techniques d'apprentissage automatique mais repose sur des principes théoriques clairement établis. Lixto assiste les utilisateurs dans la création de programmes d'extraction dans un cadre interactif. Ses créateurs soulignent toutefois que l'usage reste complexe pour un public néophyte. Des techniques d'apprentissage automatique pourraient sans doute contribuer à rendre Lixto plus accessible. Leur travail dans cette direction a commencé (Ceresna, 2005), et l'arrivée de Julien en stage postdoctoral renforcera sans doute cette direction de recherches.

Les transformations de documents XML sont souvent réalisées par un programme fonctionnant à la suite d'une extraction. C'est le cas par exemple de Lixto encore. Dans ce cas, la transformation est clairement donnée par l'utilisateur. Il n'existe pas à ma connaissance de travaux reposant sur l'apprentissage direct d'une transformation d'arbres dans le contexte d'XML. J'établis le même constat en ce qui concerne l'annotation d'arbres.

La connexion entre arbres et apprentissage automatique s'est toutefois réalisée dans quelques domaines. C'est par exemple le cas en traitement automatique des langues naturelles. Mais le propos est souvent celui de découvrir des grammaires algébriques de mots à travers la donnée ou l'utilisation de leur arbre d'analyse (Kanazawa, 1996; Sakakibara, 1992; Fernau, 2002), ou de réaliser des annotations de ces même arbres d'analyse. Le livre de Manning and Schütze (1999) donne quelques éléments introductifs en ce qui concerne une approche statistique de ces tâches. Les prolongements de recherches en inférence grammaticale se sont aussi attaqués au cas des langages d'arbres (Oncina and García, 1993; Carrasco et al., 2001; Besombes and Marion, 2002). Avec le déploiement d'Internet, de nouvelles problématiques sont apparues mêlant arbres et apprentissage comme par exemple l'inférence de DTD ou de schémas XML (Chidlovskii, 2002; Bex et al., 2006)

Enfin, d'autres approches issues de la représentation de langages par des ensembles de motifs arborescents sont apparus, parfois inscrits dans un contexte de manipulation de données XML (Goldman and Kwek, 2002; Amoth et al., 2001; Arimura et al., 2001).

# Bibliographie

Abiteboul, S., Buneman, P., and Suciu, D. (2000). *Data on the Web*. Morgan Kaufmann.

Amoth, T. R., Cull, P., and Tadepalli, P. (2001). On exact learning of unordered tree patterns. *Machine Learning*, 43(3) :211–243.

Arimura, H., Sakamoto, H., and Arikawa, S. (2001). Efficient learning of semi-structured data from queries. In *Proc. 12th International Conference on Algorithmic Learning Theory*, volume 2225 of *Lecture Notes in Artificial Intelligence*, pages 315–331.

Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual web information extraction with lixto. In *28th International Conference on Very Large Data Bases*, pages 119–128.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

Besombes, J. and Marion, J. (2002). Apprentissage des langages réguliers d'arbres et applications. In *CAP'2002*, pages 55–70.

Bex, G., Neven, F., and den Bussche, J. V. (2004). DTDs versus XML schema : A practical study. In *Proc. of WebDB*, pages 79–84.

Bex, G., Neven, F., Schwentick, T., and Tuyls, K. (2006). Inference of concise DTDs from XML data. In *Proceedings of VLDB*.

Bourret, R. (2005). XML namespaces FAQ. `http://www.rpbourret.com/xml/NamespacesFAQ.htm`.

Bruggemann-Klein, A. and Wood, D. (1998). One-unambiguous regular languages. *Information and Computation*, 140(2) :229–253.

Califf, M. (1998). Relational learning techniques for natural language information extraction. Technical Report AI98-276, IA Laboratory, university of Texas of Austin.

Carrasco, R. C., Oncina, J., and Calera-Rubio, J. (2001). Stochastic inference of regular tree languages. *Machine Learning*, 44(1/2) :185–197.

Ceresna, M. (2005). Interactive generation of html wrappers using attribute classification. In *Proceedings of the First International Workshop on Representation and Analysis of Web Space*, pages 137–142, Prague-Tocna, Czech Republic.

Chidlovskii, B. (2002). Schema extraction from xml collections. In *Proc. ACM/IEEE-CS Joint Conf. Digital Libraries*.

Clark, J. (1999). XML namespaces. `http://www.jclark.com/xml/xmlns.htm`.

Cohen, W. W., Hurst, M., and Jensen, L. S. (2002). A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the eleventh international conference on World Wide Web*, pages 232–241. ACM Press.

Crescenzi, V., Mecca, G., and Merialdo, P. (2001). Roadrunner : Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118.

Fernau, H. (2002). Learning tree languages from text. In *Proc. 15th Annual Conference on Computational Learning Theory, COLT 2002*, pages 153 – 168.

Freitag, D. and Kushmerick, N. (2000). Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583.

Goldman, S. A. and Kwek, S. S. (2002). On learning unions of pattern languages and tree patterns in the mistake bound model. *Theoretical Computer Science*, 288(2) :237 – 254.

Golgher, P. B., da Silva, A. S., Laender, A. H. F., and Ribeiro-Neto, B. (2001). Bootstrapping for example-based data extraction. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 371–378. ACM Press.

Gottlob, G. and Koch, C. (2002). Monadic queries over tree-structured data. In *17th Annual IEEE Symposium on Logic in Computer Science*, pages 189–202, Copenhagen.

Gottlob, G. and Koch, C. (2004). Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51(1) :74–113.

Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., and Flesca, S. (2004). The Lixto data extraction project - back and forth between theory and practice. In *23rd ACM SIGPLAN-SIGACT Symposium on Principles of Database Systems*, pages 1–12. ACM-Press.

Gottlob, G., Koch, C., and Pichler, R. (2003). The complexity of xpath query evaluation. In *22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 179–190.

Gottlob, G., Koch, C., and Pichler, R. (2005). Efficient algorithms for processing xpath queries. *ACM Transactions on Database Systems*, 30(2) :444–491.

Kanazawa, M. (1996). Identification in the limit of categorial grammars. *Journal of Logic, Language and Information*, 5(2) :115–155.

Kosala, R., Van Den Bussche, J., Bruynooghe, M., and Blockeel, H. (2002). Information extraction in structured documents using tree automata induction. In *6th International Conference Principles of Data Mining and Knowledge Discovery*, pages 299 – 310.

Kushmerick, N. (1997). *Wrapper Induction for Information Extraction.* PhD thesis, University of Washington.

Kushmerick, N. (2000). Wrapper induction : Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2) :15–68.

Laender, A. H. F., Ribeiro-Neto, B. A., da Silva, A. S., and Teixeira, J. S. (2002). A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2) :84–93.

Liu, L., Pu, C., and Han, W. (2000). XWRAP : An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

Muslea, I., Minton, S., and Knoblock, C. (1998). Stalker : Learning extraction rules for semistructured, web-based information sources. In *Procceeding of AAAI-98 workshop on AI and Information Extraction*.

Oncina, J. and García, P. (1993). Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93.

Poibeau, T. (2003). *Extraction automatique d'information*. Hermès, Paris.

Sahuguet, A. and Azavant, F. (2001). Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.*, 36(3) :283–316.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1) :23–60.

Soderland, S. (1999). Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3) :233–272.

# Chapitre 2

# Automates d'arbres et extraction d'informations

Je résume dans ce chapitre la contribution apportée au thème de l'extraction d'informations dans des données arborescentes. C'est principalement au cours de la thèse de Julien Carme que ce travail a été réalisé. Le premier point concerne le traitement des arbres d'arité non bornée, puis nous verrons les automates que nous avons proposés pour les manipuler. La résolution du problème d'extraction à l'aide d'automates sera ensuite exposé.

Les travaux de ce chapitre ont été réalisés avec Julien Carme et Joachim Niehren.

## 2.1 Extraction dans les arbres

Considérons un ensemble de pages Internet générées par programme contenant une information digne d'intérêt. Un exemple est donné dans la figure 2.1. On peut penser que l'ensemble des pages est construite par le serveur web, à partir d'informations contenues dans une base de données. Nous n'avons pas d'accès direct à cette base. Nous voulons toutefois obtenir l'ensemble des noms latins des oiseaux d'Europe.

Comme c'est souvent le cas, chaque donnée à extraire est ici contenue dans un noeud


Fig. 2.1 – Une page Internet contenant des données intéressantes (extrait).

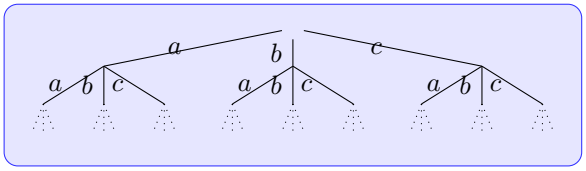



FIG. 2.2 – Arbre infini de tous les mots sur un alphabet à 3 lettres. La racine correspond au mot vide, suivre un premier fils consiste à concaténer un $a$, suivre un deuxième fils consiste à concaténer un $b$...

de l'arbre `DOM` de la page. Le problème d'extraction sera alors un problème de désignation d'un ensemble de noeuds dans un arbre. Nous garderons cette définition dans tout ce chapitre. La tâche qui nous intéresse est la construction automatique de ce programme d'extraction. C'est la dimension apprentissage automatique du problème. Nous considérons que des exemples de données à extraire seront fournis en entrée de l'algorithme d'apprentissage.

C'est une vue un peu réductrice du problème d'extraction étant donné la grande variabilité des tâches que nous avons décrites dans l'introduction de ce mémoire : nous traitons ici un problème d'extraction monadique, où chaque donnée à extraire est dans un noeud d'un arbre. Nous dépasserons un peu ces contraintes dans le chapitre suivant.

D'un point de vue fondamental, le problème d'extraction peut être formalisé de plusieurs façons. Mais si les arbres sont des termes, une formulation bien connue de ce problème peut être dégagée. Elle vient du résultat fondamental de Thatcher and Wright (1968) sur la décidabilité de la théorie monadique (faible) du second ordre à $k$ successeurs (`WSkS`).

Dans la logique `WSkS`, on dispose de $k$ opérations de successeurs qui peuvent être vues comme des lettres d'un alphabet. On dispose de variables du premier et second ordre, de quantificateurs (universels et existentiels) et des opérations logiques classiques (et, ou, non). Les variables du premier ordre sont interprétées comme des mots (finis) sur cet alphabet et celles du second ordre comme des ensembles de mots. Le problème de satisfiabilité d'une formule de cette logique est celui de trouver des interprétations pour les variables qui rendent la formule vraie. Le résultat fondamental de Thatcher propose une représentation sous forme d'arbre d'une interprétation de la formule et réduit le problème de satisfiabilité à celui du vide dans un automate d'arbres. C'est très rapidement résumé... Nous allons juste détailler les parties qui nous intéressent.

L'ensemble de tous les mots sur un alphabet à $k$ lettres peut être représenté par un arbre infini dont chaque noeud a $k$ successeurs (ou fils), un par lettre (Figure 2.2). Un noeud correspond à un mot. Une interprétation d'une variable du premier ordre est un noeud de cet arbre et celle d'une variable du second ordre est un ensemble de noeuds.

On peut alors décorer l'arbre par des étiquettes particulières pour représenter une interprétation. On prend un vecteur de booléens de taille égale au nombre de variables. Un noeud étiqueté par un vecteur $\boldsymbol{b}$ signifie que le mot correspondant est associé à ce vecteur. Le mot est dans l'interprétation de la $i$ème variable si et seulement si $b_i$, le bit à la position $i$ dans $\boldsymbol{b}$, est à 1. Toutes les interprétations ne comportent que des mots finis, l'arbre représentant une interprétation de la formule est un arbre fini (on coupe toutes les branches ne contenant que des vecteurs nuls). La figure 2.3 donne un exemple de la représentation d'une interprétation de 3 variables du second ordre valant

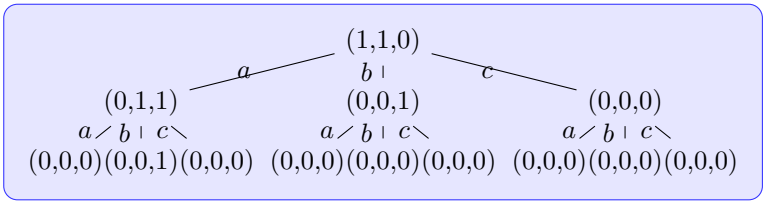



FIG. 2.3 – Un arbre représentant une interprétation de 3 variables du second ordre $X_1, X_2, X_3$. La racine est étiquetée par (1,1,0) signifie que $\epsilon$ appartient à $X_1$ et $X_2$ mais pas à $X_3$.

$X_1 = \{\epsilon\}, X_2 = \{\epsilon, a\}, X_3 = \{a, b, ab\}$.

Le résultat fondamental de Thatcher montre une équivalence entre la définissabilité[1] dans la logique `WSkS` et la reconnaissabilité par automate d'arbres. Tout ensemble d'interprétations définissable par une formule de la logique `WSkS` est reconnaissable par un automate d'arbre à travers la représentation des interprétations décrite précédemment (et vice versa). La preuve est constructive, c'est-à-dire qu'on peut exhiber cet automate étant donné une formule, mais malheureusement de complexité non élémentaire. Il faut enfin savoir que le résultat de Thatcher est très puissant : considérer des logiques plus expressives que `WSkS` provoque en général une perte de propriétés très importantes en pratique. Désigner des ensembles de noeuds, c'est donc d'un point de vue logique et dans une généralité tout à fait raisonnable, considérer une formule de `WSkS` avec des variables libres. D'un point de vue algorithmique, c'est considérer un automate fini d'arbres (finis) travaillant sur des termes d'un alphabet de vecteurs de booléens.

La dernière remarque importante dans notre cadre, est une petite astuce technique qui permet de modifier l'automate travaillant sur un alphabet d'étiquettes contenant des vecteurs de valeurs booléennes. La modification consiste à faire passer dans les états de l'automate les (ou une partie des) vecteurs de booléens. De cette façon c'est le calcul de l'automate qui « devine » les interprétations des variables. Il devine les valeurs des booléens en passant par tel ou tel état et le contrôle de la validité est conservé grâce à la condition de succès du calcul. Cette modification devient tout à fait pertinente s'il s'agit de désigner un ensemble de noeuds, par une variable libre dans une formule de `WSkS`. L'automate sera alors vu comme un moyen de « calculer » cet ensemble : une interprétation de la variable $X_i$ est obtenue en mémorisant les positions associées aux états mettant $b_i$ à 1 dans un calcul réussi de l'automate. Calculer cette interprétation signifie aussi pour notre tâche « extraire » ou encore « exécuter » si on se place dans le cadre de l'étude des requêtes sur des documents `XML`.

Au bilan, on retient de ce vieux résultat qu'on peut désigner un ensemble de noeuds dans un arbre à l'aide d'un automate d'arbres. Le calcul d'un automate d'arbres est alors une façon de calculer cet ensemble de noeuds. C'est cette approche qui est utilisée dans le travail qui suit.

La première adaptation à fournir est de traiter une des spécificités des arbres `XML`, à savoir l'arité non bornée. La notion d'arbres d'arité non bornée existe bel et bien mais n'est pas satisfaisante pour nous. Elle ne nous permet pas d'exploiter les résultats anciens que je viens de présenter. Trouver une meilleure définition de ce point de vue était un

[1] c'est-à-dire la faculté pour un (tuple de) langage(s) d'arbres d'être l'ensemble des interprétations possibles rendant une formule vraie

de nos objectifs que je décris dans la section suivante.

## 2.2 Arbres d'arité non bornée

Les recherches sur les arbres d'arité non bornée ne sont pas une nouveauté, par exemple Thatcher s'est intéressé à cette question dès les années 60. Mais depuis l'apparition des standards de l'Internet et notamment des formats `XML`, les développements technologiques ont recouru fortement à des algorithmes manipulant des structures arborescentes d'arité non bornée. C'est particulièrement le cas pour le typage ou l'analyse syntaxique de ces données. Des formalismes nouveaux ont toutefois été introduits pour manipuler les arbres d'arité non bornée. Les automates à haies reposent fortement sur les langages réguliers de mots pour désigner le langage des séquences d'étiquettes possibles sous un noeud d'un arbre `XML`. Les langages algébriques ont aussi été utilisés pour modéliser la propriété qu'un document `XML` doit être bien parenthésé. D'autres travaux ont eu recours au codage binaire fils/frère, couramment utilisé par les programmeurs pour mémoriser un arbre à l'aide de pointeurs.

À chaque fois, la question de l'expressivité s'est posée avec ces formalismes ainsi que les questions classiques de clôture et de déterminisme.

Notre approche a été de montrer que, à l'instar des travaux de Thatcher, la question de l'arité non bornée n'était pas véritablement problématique et que les automates d'arbres classiques, c'est-à-dire pour les termes, rempliraient finalement de bonnes conditions pour étudier formellement puis manipuler les structures `XML`.

Le point clef a été de reprendre la vision algébrique des automates d'arbres et d'introduire une interprétation particulière des symboles fonctionnels.

Un arbre d'arité non bornée est vu ici comme un graphe acyclique dirigé, avec une seule racine, dont chaque noeud est étiqueté par un symbole, et dont chaque noeud, hormis la racine, a exactement un seul arc entrant. Les arcs issus d'un même noeud sont totalement ordonnés.

Nous notons $\Sigma$ l'ensemble des symboles possibles qui étiquettent les noeuds. Un arbre sur $\Sigma$ peut être défini formellement par un quadruplet $(N, r, E, <, L)$ avec

- $N$ un ensemble fini dont les éléments sont les noeuds,
- la racine $r \in N$ est un noeud,
- les relations $E$ arête et $<$ sont des ordres partiels sur $N$.
- les successeurs par $E$ d'un noeud sont totalement ordonnés pour $<$. C'est-à-dire que $<$ est la une relation minimale en cardinalité qui vérifie : pour tous les noeuds $n, n_1, n_2 \in N$, si $(n, n_1)$ et $(n, n_2)$ sont deux arêtes dans $E$ alors, soit $n_1 < n_2$, soit $n_2 < n_1$.
- la racine n'est pas le successeur d'un noeud par $E$ : quel que soit $n \in N$, $(n, r) \notin E$.
- les autres noeuds ont tous un prédécesseur pour $E$ : quel que soit $n \in N$, si $n \neq r$ alors il existe un unique noeud $n' \in N$ tel que $(n', n) \in E$.
- l'étiquetage $L$ est une application de $N$ dans $\Sigma$.

Dans les technologies `XML`, les successeurs par l'ordre $E$ sont appelés fils et les successeurs par l'ordre $<$ sont appelés frères suivants.

**Notation fonctionnelle** Les arbres d'arité non bornée sont aussi notés de façon fonctionnelle. Par exemple, $a(b, b, a(c, c), c)$ désigne un arbre dont la racine étiquetée par $a$

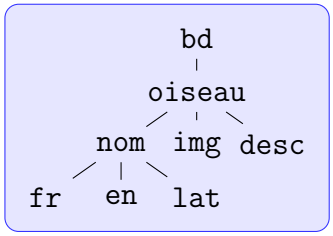



FIG. 2.4 – Arbre d'arité non bornée

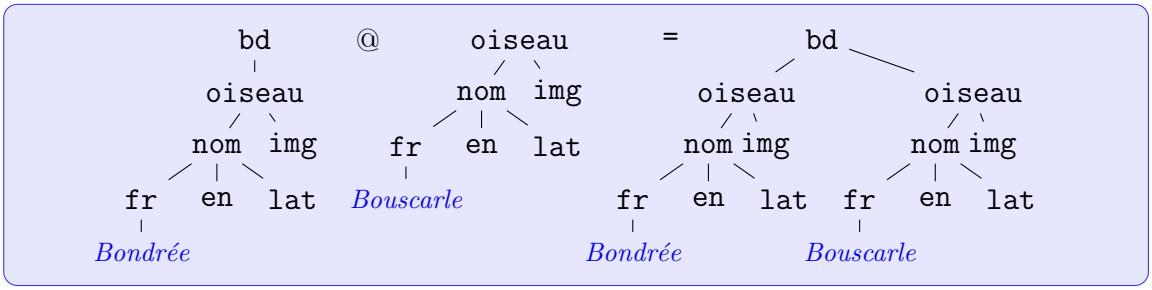



FIG. 2.5 – Adjonction d'un fils à un arbre. L'opérateur d'adjonction est noté @.

possède quatre fils étiquetés par $b$, $b$, $a$, $c$. Le troisième fils a lui même deux fils étiquetés par $c$.

**Opération d'adjonction** Puisque le nombre de fils d'un noeud n'est pas borné, il est assez naturel de munir les arbres d'une opération d'adjonction qui ajoute des fils à un arbre.

Étant donnés deux arbres $t_1$ et $t_2$, cette opération d'adjonction forme un troisième arbre $t$ en ajoutant une arête entre la racine du premier et la racine du second. La racine de $t_2$ devient le dernier fils de la racine de $t_1$. L'adjonction de $(N_2, r_2, E_2, <_2, L_2)$ à $(N_1, r_1, E_1, <_1, L_1)$ est l'arbre $(N_1 \cup N_2, r_1, E_1 \cup E_2 \cup \{(r1, r2)\}, <, L_1 \cup L_2)$. J'utilise ici un abus de notation, $L_1 \cup L_2$ désignant la fonction qui coïncide avec $L_1$ sur $E_1$ et avec $L_2$ sur $E_2$, et nous considérons bien sûr que les ensembles $E_1$ et $E_2$ sont disjoints. L'ordre $<$ inclut $<_1 \cup <_2$ mais on a également $n < r_2$ si $n$ est le dernier fils de $r_1$. Une illustration est donnée dans la figure 2.5.

**Représentation syntaxique** Il est facile de constater que cette opération d'adjonction suffit pour construire tout arbre à partir de simples arbres réduits à un noeud. Inversement, tout arbre a une décomposition unique suivant cette opération. Nous allons donc identifier un arbre et son expression (de construction à l'aide de l'adjonction) à travers une interprétation, à l'instar de ce qui est souvent fait pour les expressions arithmétiques ou booléennes (voir la figure 2.6).

L'expression d'un arbre est un terme sur $\mathcal{F}$, une signature contenant un symbole binaire noté @ et un ensemble de constantes notées $a, b, c, \ldots$. Par commodité nous employons les mêmes symboles pour les éléments de $\Sigma$ et pour les constantes de $\mathcal{F}$. Considérons maintenant l'interprétation $\mathcal{I}$ des termes de $T(\mathcal{F})$ définie selon les règles suivantes :

– Pour toute constante $a$, $\mathcal{I}(a)$ est l'arbre réduit à un noeud étiqueté par $a$ : $(\{n\}, n, \emptyset, \emptyset, L)$

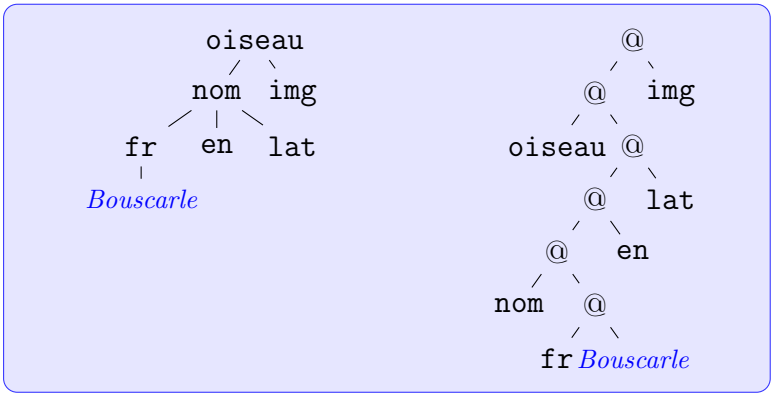



FIG. 2.6 – Un arbre d'arité non bornée et son expression : sa décomposition avec l'opérateur d'adjonction.

avec $L(n) = a$.

– $\mathcal{I}(@(t_1, t_2))$ est l'adjonction de $\mathcal{I}(t_2)$ à $\mathcal{I}(t_1)$.

Maintenant, nous pouvons constater que la fonction $\mathcal{I}$ réalise une bijection entre l'ensemble des arbres d'arité non bornée sur $\Sigma$ et l'ensemble des termes sur la signature $\Sigma$.

## 2.3 Automates pour les arbres d'arité non bornée

La mise en évidence de cet isomorphisme simple, $\mathcal{I}$, entre les algèbres de termes construits avec un symbole binaire et celle des arbres d'arité non bornée est l'élément central de cette réflexion. L'isomorphisme nous permet d'exploiter de nombreux résultats connus de longue date sur les algèbres de termes. Nous insistons ici sur le point de vue de la régularité et sur celui de la définissabilité d'ensemble d'arbres.

Les automates que nous définissons sur les arbres d'arité non bornée ne sont donc que des automates classiques sur leur représentation syntaxique binaire. C'est pour moi le côté élégant de cette approche qui n'oblige pas à introduire de nouvelles notions et de nouveaux objets. Nous obtenons alors directement la définition d'ensemble d'arbres d'arité non bornée reconnaissables. Nous obtenons aussi gratuitement toutes les opérations usuelles ainsi que les propriétés importantes dans cette classe :

– clôture par les opérations booléennes (intersection, union complémentaire) ;
– déterminisation, minimalisation.

### 2.3.1 Interprétation directe sur les arbres d'arité non bornée

Le discours précédent ne porte essentiellement que le message suivant : « *pour manipuler les arbres d'arité non bornée, oublions-les et considérons des termes binaires* ». Néanmoins, nous restons tentés d'interpréter ces automates directement sur les arbres d'arité non bornée.

L'automate peut être vu comme à l'habitude comme une machine qui va parcourir un arbre donné en entrée et mémoriser dans ce parcours des calculs intermédiaires dans des états en nombre fini.

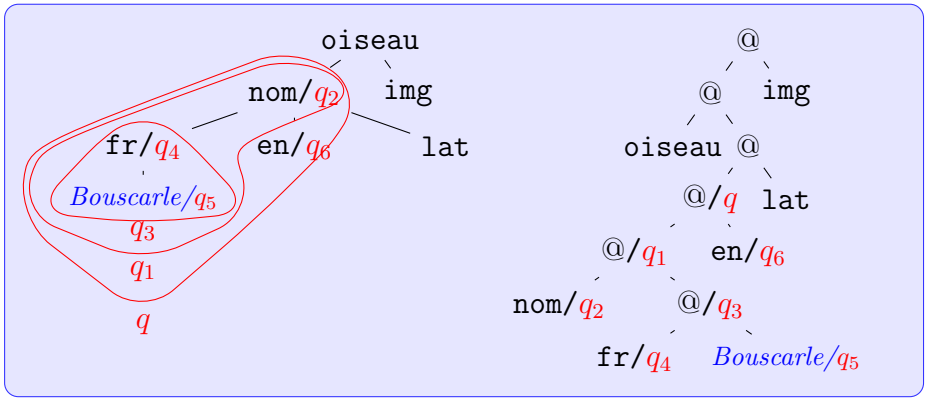



FIG. 2.7 – Interprétation directe sur les arbres d'arité non bornée. Arrivé à l'état $q$ dans l'arbre de construction, l'automate a associé un état $q$ au sous arbre d'arité non bornée entouré.

Dans cette interprétation directe nous pouvons aussi considérer deux types de parcours, correspondant aux deux parcours ascendant et descendant des automates d'arbres (*i.e.* sur les termes binaires) :

1. Dans le premier parcours, la machine commence par regarder les symboles et poursuit son calcul en construisant l'arbre de gauche à droite. La figure 2.7 donne la succession des morceaux d'arbres considérés pendant le calcul.
2. Dans l'autre parcours l'arbre est construit plutôt de la droite vers la gauche. Il commence à poser les arêtes et termine en examinant les labels des noeuds.

La première se justifie car elle est compatible avec un parcours de l'arbre en profondeur d'abord de gauche à droite. Dans le calcul, les symboles sont associés à un état dans l'ordre suivant ce parcours. Elle est alors par exemple adaptée à des traitements de flux comme dans l'approche `SAX`, même si une pile est tout de même nécessaire pour réaliser le calcul de l'automate sur un arbre.

La deuxième forme opérationnelle du calcul n'est pas interprétable de façon satisfaisante. Par contre je l'envisage éventuellement, quand l'arbre est construit en mémoire et donc dans des traitement à la `DOM`.

### 2.3.2 Relation avec les automates à haies

La notion de reconnaissabilité par automate introduite avec la représentation syntaxique de l'expression de la construction par adjonction doit être comparée avec les mécanismes de reconnaissance connus sur les arbres d'arité non bornée.

Les automates à haies ont été introduits par Murata (2000). Cette définition de la reconnaissabilité adaptée au cas de l'arité non bornée est aujourd'hui largement acceptée.

Il apparaît que les deux notions de reconnaissabilité coïncident, car elles définissent le même ensemble de langages d'arbres que l'on peut aussi qualifier de réguliers.

La définition d'automates d'arbres à haies exprime des règles pour contrôler la double récursion apparente dans les arbres d'arité non bornée. La récursion en hauteur est contrôlée comme on le fait dans les automates d'arbres classiques. La récursion en largeur sur les fils d'un noeud est, elle, contrôlée par un automate sur les séquences. L'alphabet de ces séquences est l'ensemble des états de l'automate d'arbres.

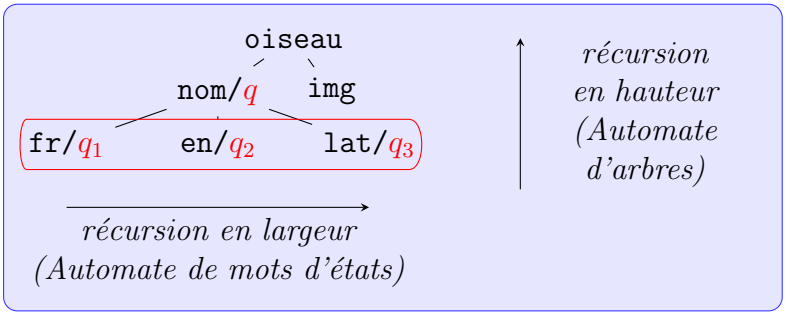



FIG. 2.8 – Automates à haies. Implante une double récursion : en hauteur avec un mécanisme proche des automates d'arbres ; en largeur avec un mécanisme d'automates de mots sur les états du premier automate.

**Définition 1.** *Un automate à haies est un quadruplet $H = (\Sigma, Q, Q_f, R)$ où $\Sigma$ est un ensemble fini de symboles, $Q$ un ensemble fini d'états, $Q_f \subseteq Q$ sont des états finaux et $R$ un ensemble de règles de transition de la forme : $a(L) \rightarrow q$ avec $L$ un langage régulier sur $Q$, $q$ un état de $Q$ et $a$ est un symbole de $\Sigma$.*

Dans une version ascendante, le calcul de $H$ commence en autorisant l'étiquetage par $q$ des symboles $a$ aux feuilles à l'aide de règles de la forme $a(L) \rightarrow q$ où $\epsilon$ appartient à $L$. Il se poursuit récursivement en autorisant l'étiquetage un noeud interne $b$ par $q'$ avec la règle $b(L') \rightarrow q'$ si ses fils $b_1, \ldots, b_n$ ont été étiquetés par $q_1, \ldots, q_n$ et $q_1 \ldots . q_n \in L'$. Une illustration est donnée dans la figure 2.8.

L'examen des automates à haies donne un autre éclairage sur notre approche. La représentation syntaxique à l'aide de termes binaires induit une reformulation de la récursion horizontale sur les fils d'un noeud de façon verticale. Elle est contrôlée à l'aide d'un automate sur les mots d'états dans l'automate à haies et à l'aide de règles d'automates d'arbres binaires dans notre cas.

Cette présence de deux mécanismes de récursion dans la définition des automates à haies est aussi l'une des raisons de la difficulté des preuves pour montrer les propriétés essentielles de ces langages : propriété de clôture, minimalisme...

**Théorème 1.** *La classe des langages reconnaissables par les automates à haies sur la signature $\Sigma$ coïncide avec la classe des images par $\mathcal{I}$ des langages de termes reconnaissables sur la signature $\mathcal{F}$.*

La preuve qui montre l'égalité des classes de langages reconnues par les deux mécanismes est assez simple. Elle est aussi constructive. Elle repose essentiellement sur la régularité des langages d'états sur les chemins père/fils dans les arbres de calcul d'un automate d'arbres.

### 2.3.3 Bilan

Les automates sur les termes peuvent via un isomorphisme représenter la classe des langages d'arbres d'arité non bornée réguliers. Bien sûr, la notion de définissabilité de langages d'arbres d'arité non bornée induite par notre définition reste identique à celle qui est connue depuis quarante ans. Il apparaît aussi que les logiques directement construites pour désigner un ensemble d'arbres d'arité non bornée sont également expressives. On

peut donc légitimement choisir la représentation des arbres d'arité non bornée via l'expression de leur construction utilisant l'adjonction. Les automates et la logique correspondante sont parfaitement connues. Le calcul d'un ensemble de noeuds par automate peut être réalisé.

En ce qui concerne l'apprentissage automatique de langages d'arbres, nous connaissons dans la littérature des algorithmes qui peuvent directement s'appliquer au cas des termes binaires.

### 2.3.4 Remarques complémentaires

Les noeuds d'un arbre d'arité non bornée sont en bijection avec les feuilles de sa représentation algébrique sous forme d'arbre binaire. Chacun des symboles @ correspond à l'ajout d'un fils. On peut aussi mettre en bijection l'ensemble des noeuds internes @ avec les arêtes de l'arbre d'arité non bornée. Le domaine complet de l'arbre, *i.e.* ses noeuds et ses arêtes est donc en bijection avec le domaine de son représentant binaire.

Le parcours en profondeur d'abord de gauche à droite d'un arbre donne une séquence de symboles. Cette séquence est aussi la séquence des feuilles de sa représentation binaire. La préservation de cet ordre naturel est une propriété importante pour les technologies `XML` et en particulier lorsqu'on considère les traitements via l'api `SAX` ou le streaming.

Le codage d'arbres d'arité non bornée en termes binaires n'est pas une nouveauté. Plusieurs codages existent et le plus répandu dans la littérature est sans doute le codage frère fils. Cependant, il ne permet pas d'obtenir avec autant de facilité une correspondance entre représentations. La difficulté provient de l'introduction de symboles $\perp$ indiquant l'absence d'un fils ou d'un frère. Il n'y a pas par exemple de bijection entre l'ensemble des noeuds ou noeuds plus arêtes de l'arbre et l'ensemble des noeuds du codage. Ceci entraîne une plus grande difficulté pour trouver un mécanisme d'automates unique sur les deux représentations. Les preuves de régularité, minimalisation et ainsi de suite s'en trouvent plus complexes.

La façon de considérer la construction d'un arbre par adjonction des fils est une opération souvent utilisée. C'est aussi de cette façon qu'on décrit une fonction $n$-naire à l'aide de fonctions binaires, en ayant recours à l'ordre supérieur. On peut voir cette opération d'adjonction comme une curryfication.

## 2.4 Réalisations

Julien a ensuite poursuivi ce travail en réalisant une application interactive qui apprend à extraire des données d'une page Internet. Le procédé repose sur l'inférence d'automates d'arbres binaires à partir d'exemples. L'algorithme utilisé, appelé Squirrel, est une adaptation de RPNI.

D'un point de vue technologique, le programme est interfacé dans Firefox sous forme d'une extension. Le résultat est assez spectaculaire puisqu'un utilisateur non informaticien construit un outil d'extraction en quelques minutes et par quelques clics. Les expériences ont aussi montré que la qualité du programme d'extraction ainsi généré est tout à fait satisfaisante en termes de précision et rappel. Il est toutefois limité à l'extraction monadique et supporte peu le bruit de structure, c'est-à-dire, des modifications légères de la structure des pages `HTML`.

J'ai ensuite voulu redéfinir l'architecture de ce programme en réalisant un découpage fonctionnel plus marqué. Le but était de le transformer en une application client-serveur, afin de permettre à plusieurs programmes d'apprentissage de fonctionner et de diversifier son utilisation, en laissant par exemple le choix de l'interface utilisateur. Je voulais aussi standardiser le format des documents de façon à disposer de jeux de données en grand nombre exploitables par l'équipe toute entière et par la communauté. Le serveur a été développé par Missi Tran et est maintenant accessible via Internet. Différents clients on été développés par plusieurs étudiants. Dans un avenir proche, un client sous forme de Web service sera opérationnel. Plusieurs programmes d'apprentissage sont aujourd'hui intégrés à cette plateforme.

## 2.5 Conclusion

La thèse de Julien Carme a été la première thèse soutenue dans Mostrare. Julien a su travailler avec de nombreux membres de l'équipe et il a su parfaitement prendre son autonomie au bon moment. Il est actuellement accueilli dans l'équipe Lixto à Vienne.

Pour moi, elle a aussi montré que les objectifs définis dans le projet Mostrare étaient tout à fait pertinents. Elle a aussi permis de poursuivre nos efforts dans les directions suivantes :

- Représentation des données : Réaliser des automates sur des arbres `XML` demande d'abord de réaliser une abstraction de ces arbres. En effet, les automates travaillent sur un alphabet fini de symboles. Hors, les feuilles contenant du texte, qui représentent souvent la partie à extraire, doivent être typées pour rendre fini l'ensemble des valeurs possibles. Dans le même esprit, il convient de limiter la taille de cet alphabet, sans quoi l'apprentissage n'est plus possible.
- Le cas des relations $n$-aires. Les problèmes d'extraction que nous avons considérés sont des problèmes monadiques. D'un point de vue théorique, la démarche présentée ici fonctionne encore dans le cas $n$-aire, en considérant $n$ variables libres. Il est par contre nécessaire de choisir une représentation dans les arbres des occurrences de la relation de façon à éviter de perdre l'association entre les composantes. Une façon canonique de procéder est de ne permettre à toute variable libre de ne prendre qu'une seule valeur dans toute interprétation. On encode de cette façon une seule occurrence de tuple par arbre. Par contre, cette approche peut être critiquée du point de vue de l'apprentissage pour des raisons de complexité.
- L'expressivité. Grâce à cette relation avec la logique `WSkS` et les automates d'arbres, la puissance d'expression des outils d'extraction est clairement identifiée. Par contre, nous pouvons exhiber des cas pratiques où les langages d'arbres ne sont plus réguliers parce qu'il est nécessaire de compter pour localiser l'information à extraire. C'est le cas par exemple de colonnes de tableaux.

## 2.6 Notes bibliographiques

Dans les années 60 et 70, plusieurs chercheurs remarquables ont posé les bases de la théorie des langages formels d'arbres. C'est sans doute Thatcher (1967) qui a pour la première fois donné une définition de reconnaissabilité de langages d'arbres d'arité non

bornée, alors appelés pseudo-termes. Plusieurs auteurs les ont aussi étudiés en proposant des caractérisations différentes (Brainerd, 1969; Pair and Quere, 1968). L'expansion des langages SGML et XML ont replacé les arbres d'arité non bornée et leur automates dans un contexte nouveau. La caractérisation par des arbres de dérivation de grammaires régulières étendues a donné lieu à des algorithmes d'analyse syntaxique efficaces pour SGML (Brüggemann-Klein and Wood, 1998) et s'est aussi appliquée à XML (Murata, 2000; Brüggemann-Klein et al., 2001). Elle a donné lieu aux automates à haies.

Les travaux des années 60 sur les automates d'arbres ont aussi abouti à la décidabilité de la logique WSkS par Thatcher and Wright (1968). Deux articles de Thomas (1990, 1997) donnent une explication remarquable des preuves que l'on retrouvent aussi dans Comon et al. (1997, chap 3).

Très rapidement, l'utilisation d'automates pour l'évaluation de requêtes dans les arbres XML a été envisagée. La connexion de ce calcul avec la logique monadique a aussi fait l'objet de résultats notamment par Neven (1999); Neven and Schwentick (1999, 2002) et Gottlob and Koch (2002b, 2004, 2002a). La modélisation des arbres est ici plus proche des automates à haies. Elle demande alors de montrer des propriétés (comme la définissabilité par exemple) avec de nombreux efforts techniques. Le passage à une caractérisation par un codage en arbres binaires est aussi présent dans le travail de Neven (1999) et dans Neumann and Seidl (1998). Mais le codage envisagé est le codage frère fils qui ne donne pas immédiatement de correspondance entre automates pour arbres d'arité non bornée et automates d'arbres binaires. Dans (Cristau et al., 2005), le problème de la minimalisation des automates d'arbres d'arité non bornée est étudié. Dans ce sujet, la contribution de Martens and Niehren (2006) vient appuyer encore l'avantage de notre approche algébrique en comparaison avec d'autres méthodes directes sur les arbres d'arité non bornée.

L'algorithme d'apprentissage de langages d'arbres d'arité non bornée utilisé par Julien dans son travail avec Squirrel dérive de RPNI (Oncina and Garcia, 1992). C'est un travail d'abord formulé pour les séquences mais qui repose sur des principes que l'on retrouve à la fois dans les automates de mots et dans les automates d'arbres. Il donne un algorithme polynomial pour l'identification d'un langage représenté par automate fini déterministe à partir d'exemples et de contre-exemples du langage cible. La nécessité de contre-exemples se traduit par la contrainte de disposer d'arbres totalement annotés par ces informations booléennes aux noeuds (et même d'arbres peut-être uniquement annotés de valeurs 0). L'adaptation de Carme et al. (2006) consiste essentiellement à relâcher partiellement cette contrainte. Les exemples négatifs sont en quelque sorte obtenus en considérant une heuristique basée sur un élagage « glouton ». Se passer totalement d'exemples négatifs, tout en gardant un résultat d'apprenabilité au sens de Gold, demande certainement de se restreindre à des classes de langages trop peu expressives pour notre tâche d'extraction (comme par exemple les langages réversibles d'arbres (Besombes and Marion, 2002)). Pour le cas de l'extraction $n$-aire, une extension de cet algorithme a été présentée dans Lemay et al. (2006).

En extraction d'informations, plusieurs travaux reposent sur de l'inférence grammaticale. On peut citer Chidlovskii (2001); Chidlovskii et al. (2000); Chidlovskii (2000) utilisant des distances d'édition ou des algorithmes pour les langages $k$-réversibles ou encore Lerman et al. (2001) utilisant ALERGIA de Carrasco and Oncina (1994). Des publications très proches de notre angle d'attaque du problème d'extraction sont dues à l'équipe de Maurice Bruynooghe en Belgique (Kosala et al., 2002, 2003; Raeymaekers

et al., 2005).

# Bibliographie

Besombes, J. and Marion, J. (2002). Apprentissage des langages réguliers d'arbres et applications. In *CAP'2002*, pages 55–70.

Brainerd, W. S. (1969). Tree generating regular systems. *Information and Control*, 14(2) :217–231.

Brüggemann-Klein, A. and Wood, D. (1998). One-unambiguous regular languages. *Inf. Comput.*, 140(2) :229–253.

Brüggemann-Klein, A., Wood, D., and Murata, M. (2001). Regular tree and regular hedge languages over unranked alphabets : Version 1.

Carme, J., Gilleron, R., Lemay, A., and Niehren, J. (2006). Interactive learning of node selecting tree transducers. *Machine Learning*.

Carrasco, R. and Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *International Conference on Grammatical Inference*, pages 139–152, Heidelberg. Springer-Verlag.

Chidlovskii, B. (2000). Wrapper generation by k-reversible grammar induction. In *ECAI'00 Machine Learning for Information Extraction Workshop*.

Chidlovskii, B. (2001). Wrapping web information providers by transducer induction. In *Proc. European Conference on Machine Learning*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 61 – 73.

Chidlovskii, B., Ragetli, J., and de Rijke, M. (2000). Wrapper generation via grammar induction. In *Proceedings of the 11th European Conference on Machine Learning*, pages 96–108.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. Available on : http ://www.grappa.univ-lille3.fr/tata.

Cristau, J., Löding, C., and Thomas, W. (2005). Deterministic automata on unranked trees. In *15th International Symposium on Fundamentals of Computation Theory*. To Appear.

Gottlob, G. and Koch, C. (2002a). Monadic datalog and the expressive power of languages for web information extraction. In *21rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 17–28. ACM-Press.

Gottlob, G. and Koch, C. (2002b). Monadic queries over tree-structured data. In *17th Annual IEEE Symposium on Logic in Computer Science*, pages 189–202, Copenhagen.

Gottlob, G. and Koch, C. (2004). Monadic datalog and the expressive power of languages for web information extraction. *Journal of the ACM*, 51(1) :74–113.

Kosala, R., Bruynooghe, M., Van den Bussche, J., and Blockeel, H. (2003). Information extraction from web documents based on local unranked tree automaton inference. In *18th International Joint Conference on Artificial Intelligence*, pages 403–408. Morgan Kaufmann.

Kosala, R., Van Den Bussche, J., Bruynooghe, M., and Blockeel, H. (2002). Information extraction in structured documents using tree automata induction. In *6th International Conference Principles of Data Mining and Knowledge Discovery*, pages 299 – 310.

Lemay, A., Niehren, J., and Gilleron, R. (2006). Learning n-ary node selecting tree transducers from completely annotated examples. In *International Colloquium on Grammatical Inference*, volume 4201 of *Lecture Notes in Artificial Intelligence*, pages 253–267. Springer Verlag.

Lerman, K., Knoblock, C. A., and Minton, S. (2001). Automatic data extraction from lists and tables in web sources. In *Proceedings of Automatic Text Extraction and Mining workshop (ATEM-01), IJCAI-01.*

Martens, W. and Niehren, J. (2006). On the minimization of XML schemas and tree automata for unranked trees. *Journal of Computer and System Science.*

Murata, M. (2000). Hedge automata : a formal model for xml schemata. Web page.

Neumann, A. and Seidl, H. (1998). Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145.

Neven, F. (1999). *Design and Analysis of Query Languages for Structured Documents. Aformal and logical Approach.* PhD thesis, Limburgs Univ.

Neven, F. and Schwentick, T. (1999). Query automata. In *Proceedings of the Eighteenth ACM Symposium on Principles of Database Systems*, pages 205–214.

Neven, F. and Schwentick, T. (2002). Query automata over finite trees. *Theoretical Computer Science*, 275(1-2) :633–674.

Oncina, J. and Garcia, P. (1992). Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pages 49–61.

Pair, C. and Quere, A. (1968). Définition et études des bilangages réguliers. *Information and Control*, 13(6) :565–593.

Raeymaekers, S., Bruynooghe, M., and Van den Bussche, J. (2005). Learning (k,l)-contextual tree languages for information extraction. In *Proceedings of ECML'2005*, volume 3720 of *Lecture Notes in Artificial Intelligence*, pages 305–316.

Thatcher, J. W. (1967). Characterizing derivation trees of context-free grammars through a generalization of automata theory. *Journal of Computer and System Science*, 1 :317–322.

Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2 :57–82.

Thomas, W. (1990). *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. MIT Press.

Thomas, W. (1997). Languages, automata and logic. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3, pages 389–456. Springer Verlag.

# Chapitre 3

# Extraction dans les arbres et classification supervisée

Je résume dans ce chapitre le travail réalisé dans le cadre de la thèse de Patrick Marty. Le fil directeur consiste à voir le problème de l'extraction comme un problème de classification. Nous avons attaqué les problèmes d'extraction $n$-aire à l'aide de cette méthode. Nous avons obtenu un algorithme interactif pour la génération d'outils d'extraction de relations $n$-aires. L'algorithme intègre une dimension statistique qui donne un peu plus de robustesse par rapport à des méthodes purement syntaxiques.

Rémi Gilleron, Fabien Torre et bien sûr Patrick Marty sont associés à ce travail.

Je présente le cadre général de la réduction de la tâche d'extraction à un problème de classification, en détaillant les choix de la représentation des données depuis les séquences jusqu'aux arbres `XML`. Le passage à l'extraction de relations puis au cadre interactif est abordé dans la section 3.2.

## 3.1 Classer pour extraire

Dans le chapitre précédent, nous avons vu qu'une représentation adéquate des données a permis de bâtir un algorithme d'apprentissage de programmes d'extraction. En résumé, les arbres ont été décorés par une étiquette booléenne « à extraire » ou « à ne pas extraire » et on apprend un automate d'arbres capable de reconnaître le langage des arbres annotés des bonnes étiquettes à chaque noeud. Nous ne manipulons pas de langages d'arbres dans ce chapitre, mais c'est avec des algorithmes de classification de noeuds que nous voulons calculer les bonnes étiquettes.

L'inférence de classifieurs à partir d'exemples est un champ de recherches ancien et toujours actif dans la communauté apprentissage automatique. Nous voulions d'abord bénéficier de cette expertise. Le deuxième avantage repose sur la nature statistique des algorithmes d'inférence de classifieurs. Le troisième avantage est de mieux contrôler la représentation des données et pouvoir intégrer des connaissances du domaine.

Dans cette démarche, on constate une grande sensibilité au choix de la représentation des données. Patrick a commencé à travailler sur les textes pour étudier l'incidence de la représentation des données sur la qualité des programmes d'extraction appris. Pour une tâche d'extraction, il faut dans un premier temps découper le texte en unités lexicales. Ce sont ces unités que nous allons classer. Le choix de ce découpage est un premier

paramètre. Ensuite, nous distinguons deux étapes : la phase d'apprentissage et la phase d'extraction. La figure 3.1 décrit le fonctionnement général et les deux phases dans le cas d'une extraction monadique.

La phase d'apprentissage est réalisée lors de la conception du programme d'extraction. On dispose de documents annotés, c'est-à-dire où les informations à extraire sont indiquées. À l'aide d'un analyseur lexical, le texte est découpé. Puis la représentation de chaque unité lexicale est calculée. La représentation inclut l'information dite « de classe » à savoir, si l'unité doit être extraite ou non. L'algorithme d'apprentissage proprement dit est alors utilisé avec ces données en entrée pour bâtir un classifieur (ou procédure de classification). Le classifieur est une procédure capable de calculer une classe à partir d'une représentation.

La phase d'extraction est utilisée en ligne. L'analyse lexicale et le calcul de la représentation sont communes à l'apprentissage, à la différence que l'information de classe n'est pas présente. C'est le classifieur induit qui la déterminera.

### 3.1.1 Représentation attributs/valeurs

Les algorithmes de classification courants fonctionnent à partir d'une représentation en attribut/valeur. Ici, chaque unité lexicale sera donc représentée par un enregistrement. Par exemple, on pourra considérer dans cette représentation les attributs suivants : sa taille en nombre de symboles, si elle commence par une majuscule, si elle n'est écrite qu'avec des chiffres... L'ensemble des enregistrements ainsi constitués, accompagnés de leur classe, c'est-à-dire l'information s'ils doivent être extraits ou pas est donnée à un algorithme de classification supervisée. Nous avons souvent utilisé C5 qui génère des arbres de décision. Le modèle de classification servira ensuite pour la tâche d'extraction.

Lors de la phase d'extraction, le système prend en entrée un document, applique le découpage en unités lexicales, calcule la représentation de ces unités en attribut/valeur et les classe.

La première constatation est que la représentation en un ensemble d'enregistrements de tout document doit se faire rapidement pour obtenir une procédure efficace. Nous avons privilégié des attributs dont les valeurs peuvent être efficacement calculées au chargement des documents. De telle sorte, ce pré-traitement est finalement une opération peu coûteuse. Il faut noter que quelle que soit la méthode choisie pour réaliser la tâche d'extraction d'informations, un pré-traitement similaire est toujours obligatoire, même dans le cas du chapitre précédent, mais il est souvent occulté.

Opter pour une représentation en attributs/valeur permet aussi d'insérer des connaissances du domaine dans cette représentation et ceci très facilement. Par exemple, sachant qu'on extrait des noms latins d'oiseaux, on peut tester la terminaison des mots et ajouter un attribut booléen indiquant que le mot se termine en *a*, *us*, *or*...

### 3.1.2 Des textes aux arbres

Lorsque les données ne sont plus des textes mais des arbres, la représentation peut être enrichie. D'abord le découpage en unités lexicales est partiellement donné par le découpage en feuilles. On peut se contenter du découpage en feuilles si le document est très structuré, que les parties à extraire correspondent à des feuilles ou des noeuds.

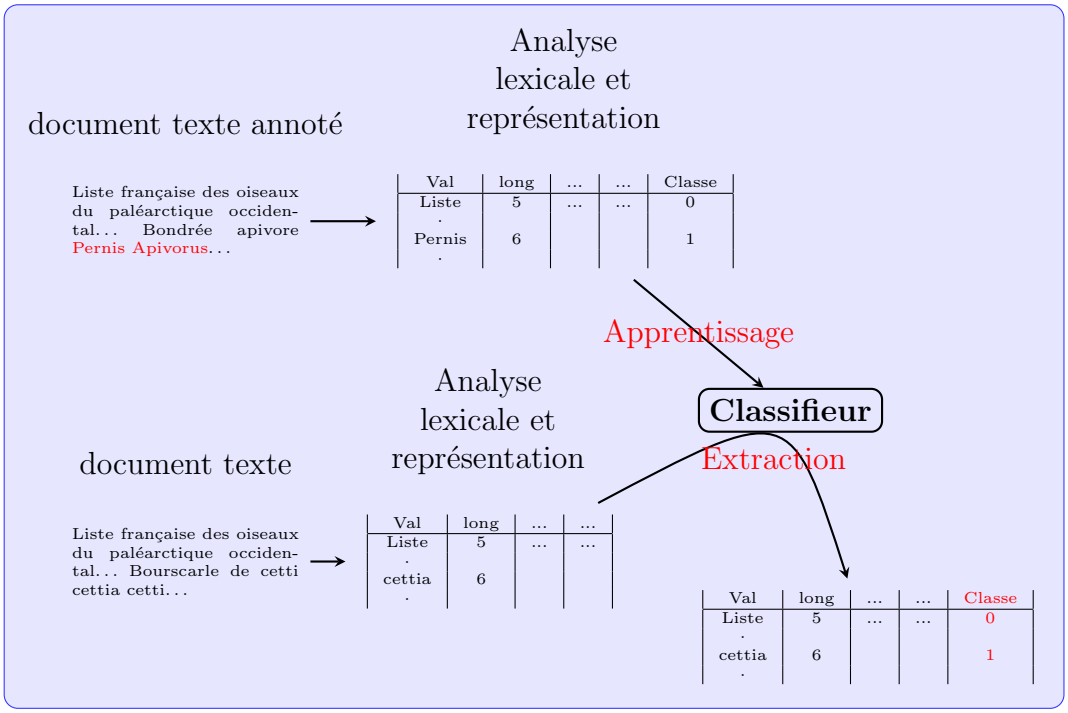



FIG. 3.1 – Architecture générale

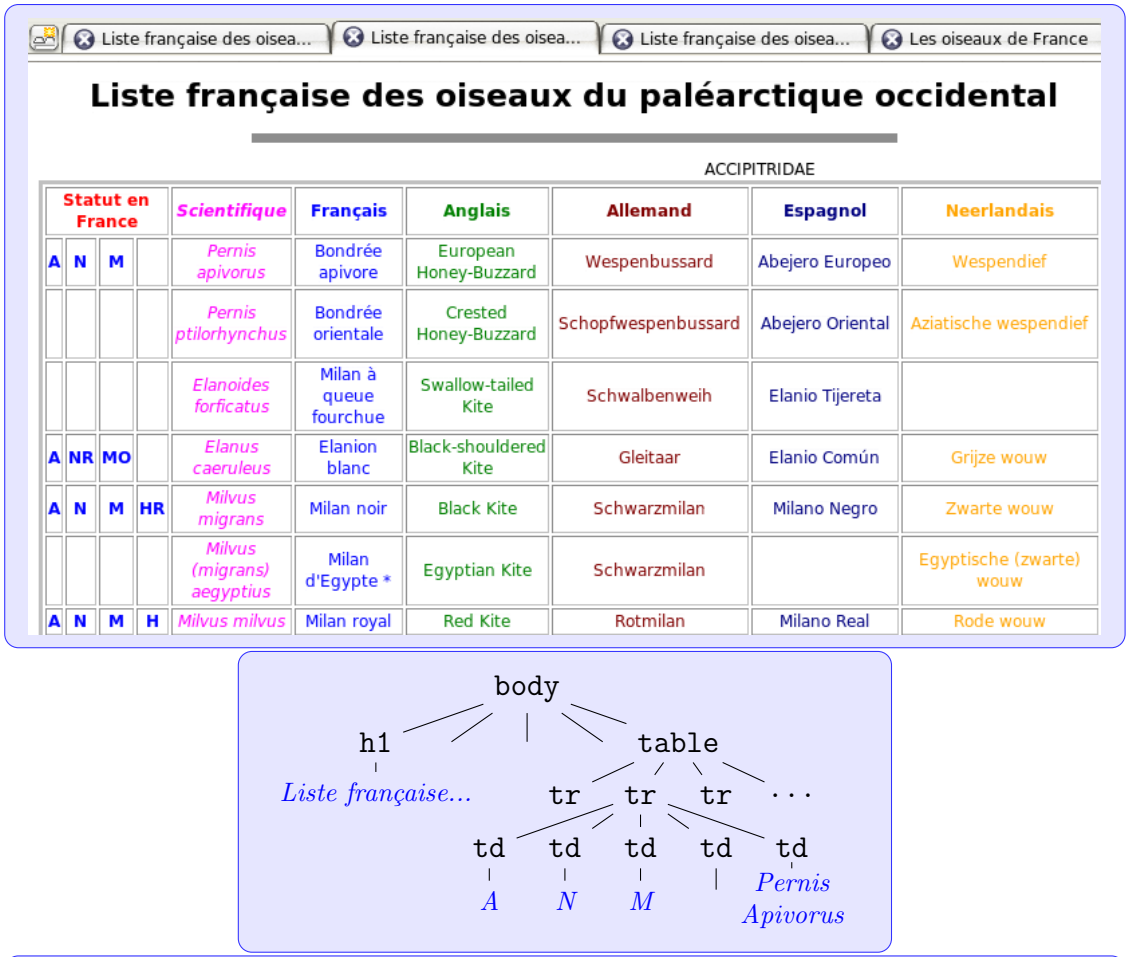
Ensuite, des informations sont induites de la structure arborescente de la donnée. Par exemple : quel est le symbole du père ? ou quel est le numéro dans la fratrie ? En fait, plusieurs vues du document sont possibles lorsque l'on parle d'un document `XML`. Elles sont illustrées dans la figure 3.2. Chacune va enrichir la représentation des unités lexicales. La vue `DOM` est celle de l'arbre `XML`, la vue texte est la concaténation des feuilles textes dans un parcours en profondeur d'abord de gauche à droite. La vue fichier (ou flux) correspond à la sérialisation du document sous la forme d'une séquence contenant les balises et le texte. On pourrait ajouter une vue en deux dimensions obtenue à partir du rendu dans un navigateur. Mais cela n'a de sens que pour les documents susceptibles d'avoir un rendu comme le `XHTML` ou du `XML` associé à une feuille de style. Le rendu est de plus très dépendant du terminal et du média utilisé. Cette vue prend son sens selon moi pour des documents de rendu fixé, destinés par exemple à l'impression comme le `PDF`.

### 3.1.3 Du monadique au $n$-aire

L'extraction $n$-aire consiste à trouver les occurrences d'une relation dans un document. C'est par exemple extraire les couples des noms en français et en latin. Chaque nom (latin et français) correspondant à une **composante** de cette relation.

Est-ce que se contenter de problèmes d'extraction monadiques est réellement une limitation ? La première idée est d'appliquer $n$ extractions, pour chacune des composantes et ré-associer les composantes ensuite. Est-ce que la ré-association est facile ?

Pour répondre à cette question, nous avons d'abord étudié comment une relation $n$-aire était souvent stockée dans un arbre. Un bon angle d'attaque pour cette question était celui des outils et librairies capables de produire des documents `XML` à partir de tables




Liste française des oiseaux du paléarctique occidental Accipitridae Statut en France Scientifique Français... A N M Pernis Apivorus Bondrée apivore...

```
<HTML> <HEAD> <meta http-equiv="content-type" content="text/html ;
charset=utf-8"> <SCRIPT> <TITLE> Liste française des oiseaux du pa-
léartique occidental </TITLE> </HEAD> <BODY BGCOLOR="#ffffff"> <H1
ALIGN="center"> Liste française des oiseaux du paléarctique occi-
dental </H1>
<HR ALIGN="center" SIZE="7" WIDTH="60
<TABLE BORDER="3"> <CAPTION> ACCIPITRIDAE </CAPTION> <TBODY> <TR
ALIGN="center"> <TH COLSPAN="4"> <FONT COLOR="#FF0000"> Statut en
France </FONT>
</TH> <TH> <I> <FONT COLOR="#FF00FF"> Scientifique </FONT> </I> ...
```

FIG. 3.2 – Vues d'un document XML : Le rendu, l'arbre DOM, la vue texte et la vue fichier.

d'une base de données relationnelle. Il suffit de regarder par exemple les données `XML` produites par un tableur comme OpenOffice, des tableaux de bord `HTML` produits par des générateurs comme JasperReport ou Business Object. Nous avons identifié cinq façons d'organiser des $n$-uplets issus d'une même relation dans un arbre. Nous les considérons comme cinq cas de base, illustrés dans la figure 3.3 :

1. Les listes où les composantes d'un même $n$-uplet apparaissent consécutivement, toujours dans le même ordre. C'est ce que l'on retrouve dans les résultats de moteur de recherche simples. L'arbre correspondant est souvent très plat et très large.
2. Les tables où les $n$-uplets sont rangés par ligne. Là encore, les composantes d'un même $n$-uplet apparaissent consécutivement, toujours dans le même ordre. Souvent la structure arborescente est un peu plus profonde et les composantes sont rassemblées sous des noeuds identifiant chaque $n$-uplet. Les listes imbriquées suivent aussi cette organisation.
3. Les tables tournées où les $n$-uplets sont rangés par colonne. C'est souvent le cas des relations avec un grand nombre de composantes. Dans ce cas les composantes des $n$-uplets ne sont pas rassemblées sous des noeuds deux à deux distincts. Cette fois les composantes d'un même $n$-uplet ne sont plus consécutives dans l'arbre. Elles alternent d'abord $n$-uplet par $n$-uplet, puis composante par composante.
4. Les structures factorisées ne répètent pas les même valeurs. On obtient des structures imbriquées par valeur. Par exemple une liste de villes par pays. Une occurrence d'une valeur peut donc appartenir à plusieurs $n$-uplets.
5. Les tables croisées organisent les $n$-uplets à la fois en ligne et en colonne. L'exemple le plus simple est la table des distances entre villes. Mais on retrouve aussi naturellement cette organisation dans les tableaux de bord ou deux factorisations ont été opérées et placées en lignes et en colonne : par exemple les ventes par région et par produit.

Nous avons retrouvé ces cinq cas dans de nombreux sites Internet et composés les uns avec les autres. Il apparaît alors que la ré-association des $n$-uplets à partir des extractions unaires est une opération qui ne peut se faire automatiquement. La seule possibilité est de réaliser l'extraction en même temps que la ré-association pour exploiter l'information complète contenue dans le document. C'est ce que réalise l'algorithme PAF.

## 3.2 Extraction de relations dans les arbres

PAF est un algorithme d'extraction de relations $n$-aires qui construit la relation cible à partir de classifieurs binaires. Deux idées de base gouvernent son fonctionnement : l'enrichissement de la représentation et l'incrémentalité.

PAF fonctionne incrémentalement en construisant des classifieurs binaires de $n$-uplets de longueur croissante dans une boucle de 1 jusque $n$.

Chaque $n$-uplet de longueur $i$ est représenté dans un codage attribut valeur et cette représentation est adaptée à chaque itération de la boucle. De ce fait, il est possible d'intégrer à l'étape $i$ des informations tenant compte des résultats de l'extraction à l'étape $i-1$. C'est l'idée de l'enrichissement de la représentation.

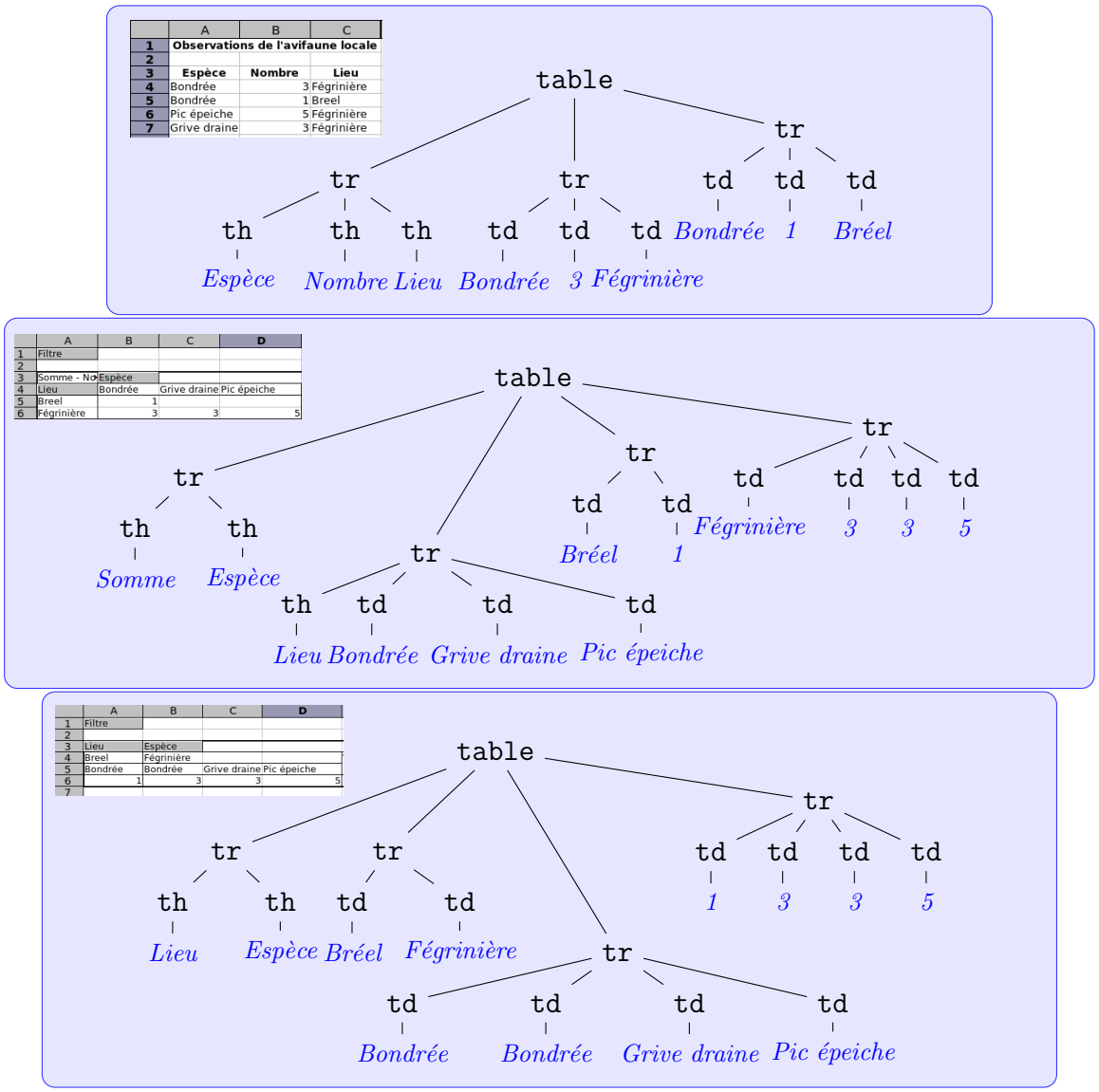



Fig. 3.3 – Différentes organisations et leur arbre correspondant. En haut une simple table ; au milieu une table croisée ; en bas une table tournée avec factorisation : les enregistrements sont en colonne et la valeur Fégrinière est factorisée pour trois enregistrements.

### 3.2.1 Incrémentalité

#### Extraction

L'algorithme d'extraction est incrémental. Il est paramétré par $n$ classifieurs et reçoit un document en entrée. Il extrait la première composante. C'est un problème monadique qui est réalisé comme indiqué dans la section 3.1 avec le classifieur $c_1$. Les résultats de cette extraction sont injectés dans la représentation de couples de composantes. C'est cette information supplémentaire qui nous aide à recomposer les composantes entre-elles pour former les $n$-uplets.

Le danger immédiat de cette approche est combinatoire. Si le document comporte $k$ unités susceptibles d'être extraites, on classe $k$ candidats à la première étape, mais $k^2$ à la deuxième puisqu'on forme alors des couples de candidats, et $k^3$ à la troisième.... Pour éviter cette exponentielle en la longueur de la relation, nous avons fait le choix suivant. Les candidats à l'étape $i$ sont les $n$-uplets extraits à l'étape $i-1$ augmentés d'une composante. Le nombre de candidats devient donc linéaire à chaque étape.

La réduction de ce problème combinatoire a des conséquences sur l'ordre dans lequel les extractions sont réalisées. Si le premier problème monadique est mal traité, on perd toute chance d'extraire les bons $n$-uplets. Nous avons envisagé d'utiliser des techniques d'auto-évaluation classiques en apprentissage pour définir l'ordre d'extraction.

#### Apprentissage

L'algorithme d'apprentissage consiste à apprendre un classifieur de $n$-uplets. Le processus d'apprentissage est incrémental à l'image de l'algorithme d'extraction. On apprend un classifieur pour la première composante, puis les couples,....

Les apprenants de classifieurs binaires ont en entrée un ensemble d'exemples positifs et un ensemble d'exemples négatifs. À l'étape $i$ du processus incrémental, chaque exemple est la représentation d'un $n$-uplet de longueur $i$. Comme pour l'extraction, le nombre d'exemples, essentiellement pour les exemples négatifs, peut croître exponentiellement avec $n$. Nous employons le même procédé pour garder un équilibre de classes qui mettrait l'apprentissage en difficulté.

Nous avons utilisé un apprenant fonctionnant à partir d'exemples positifs et négatifs. Nous devons donc supposer que des exemples négatifs sont fournis mais il n'est pas satisfaisant de demander à l'utilisateur des exemples de ce qu'il ne veut pas extraire. Nous avons plusieurs solutions pour contourner ce problème. La première est de demander d'annoter tous les exemples positifs d'un document et nous obtenons les négatifs par différence. L'annotation peut être longue dans le cas de grands documents. La deuxième est de d'annoter complètement une partie identifiée. La troisième est d'appliquer des heuristiques qui consistent généralement à identifier une partie où l'annotation a été faite complètement.

Cette troisième solution est bien adaptée dans le cas de l'extraction monadique. Elle a été mise en place par Julien dans sa thèse par exemple. Mais dans le cas $n$-aire, à cause des organisations potentiellement complexes des données décrites en section 3.1.3, les heuristiques résistent mal. Une quatrième approche, que nous avons utilisée, a été de trouver des heuristiques permettant d'obtenir des exemples négatifs.

Une alternative aurait été de procéder avec un algorithme de classification ne fonc-

tionnant qu'avec des exemples positifs. Nous ne l'avons pas (encore) testée[1].

### 3.2.2 Enrichissement de la représentation

Les représentations de chaque $n$-uplet exemple sont fixées. Elles sont basées sur des représentations de noeuds (noeuds internes et feuilles).

**Représentation des noeuds** Elle inclut l'étiquette, la position du noeud dans la fratrie, la hauteur, la profondeur, les étiquettes des noeuds voisins, la taille du sous-arbres sous le noeud. Dans le cas du `XHTML`, elle inclut la valeur des attributs `class`.

**Représentation des feuilles** Elle étend la représentation des noeuds par la représentation de ses 5 ancêtres dans la vue `DOM` et la représentation des textes précédents et suivants dans la vue texte. On utilise au total environ 60 attributs.

L'enrichissement d'une étape $i$ à $i+1$ est réalisé en mémorisant seulement deux couples de noeuds : celui de la composante 1 et de la composante $i+1$ et celui de la composante $i$ et la composante $i+1$.

**Représentation des couples** Pour une paire de noeuds $(p, m)$ les attributs sont la représentation du plus petit ancêtre commun $a$, les longueurs des plus courts chemins entre $p$, $m$ et $a$ deux à deux, le nombre de feuilles texte entre $p$ et $m$,enfin sur une fenêtre de 5 ancêtres, les différences entre les positions des noeuds ancêtres de $p$ et $m$ dans la séquence des fils de leur père, ce qui donne environ 20 attributs. Ces derniers attributs sont typiquement utiles pour traiter les enregistrements rangés par colonne dans les tables tournées.

De ce fait, tous les $n$-uplets sont décrits avec des enregistrements de même taille. Le calcul de cette représentation est linéaire pour les descriptions de noeuds et feuilles et quadratique pour les couples. Les attributs exploitent soit la vue `DOM` soit la vue texte des données.

### 3.2.3 Cadre interactif

L'intérêt de replacer notre algorithme dans un cadre interactif est de fournir un programme à destination d'utilisateurs finaux à l'instar de ce qui a été réalisé par Julien dans sa thèse. La dimension interactive doit permettre de réduire le nombre d'exemples à étiqueter pour apprendre et donc limiter le travail de l'utilisateur final. Ce passage est aussi une voie ouverte sur des idées d'apprentissage actif. Un algorithme d'apprentissage actif interagit fortement avec l'utilisateur en l'interrogeant, par exemple en lui proposant des exemples à étiqueter. Cette capacité peut réduire fortement le nombre d'interactions si par exemple les « bons » exemples sont choisis. C'est un champ de recherches qui reste à développer dans notre équipe.

Dans un cadre interactif, les informations ne sont pas données par l'intermédiaire d'un ensemble annoté. Cet ensemble se construit de façon interactive par l'ajout d'exemples positifs ou négatifs après chaque hypothèse proposée par l'apprenant. Nous modélisons cette interaction par un oracle qui renvoie vrai si l'hypothèse est correcte ou renvoie

[1] J'ai réalisé avec François Denis et Rémi Gilleron un tel algorithme. Il demande de connaître ou estimer la densité des positifs. Il est rapidement décrit dans le chapitre 5.

un exemple annoté contredisant l'hypothèse courante dans le cas contraire. Dans l'algorithme d'apprentissage, en plus de la boucle sur les composantes, une deuxième boucle est ainsi constituée modélisant la répétition des interactions jusqu'à l'obtention d'une hypothèse correcte. Deux questions doivent alors être discutées : la prise en compte de la notion de document dans le processus et l'imbrication des deux boucles.

Nous avons choisi que les interactions se feraient composante par composante, et donc que la boucle sur les interactions est imbriquée dans celle des composantes. Ce choix est justifié par le fait que nous faisons l'hypothèse que la disposition des $n$-uplets dans le document est complexe et donc que la désignation d'un $n$-uplet entier n'est pas aisée. Par exemple, dans le cas de structures avec factorisations, il apparaît plus économique en interactions de commencer par désigner les composantes factorisées d'un $n$-uplet plutôt que d'énumérer tous ses développements.

L'utilisateur est modélisé par un oracle qui retourne soit vrai soit un couple $(e, d)$ formé d'un exemple $e$ dans un document $d$. Nous supposons que par souci d'économie de la part de l'utilisateur, les interactions se déclinent document par document. En conséquence, nous supposons que lorsque l'utilisateur change de document, il considère que le document courant est correctement et complètement annoté. Aussi, au cours de l'apprentissage, le programme manipule un ensemble de documents étiquetés ainsi qu'un document courant partiellement étiqueté.

Le traitement des exemples négatifs est différent dans les deux cas. Pour l'ensemble d'exemples totalement étiquetés, il paraît nécessaire de limiter le nombre d'exemples négatifs. Par contre, en présence de documents partiellement étiquetés, nous ne connaissons que les indications fournies par l'utilisateur et nous sommes donc en présence d'un très grand nombre d'exemples non étiquetés. Il devient nécessaire d'essayer d'obtenir le maximum d'informations des interactions (surtout pour les premières interactions, quand on ne dispose pas du tout de document étiqueté).

C'est ici que nous mettons en place des heuristiques qui permettent d'induire des exemples négatifs à partir de l'annotation partielle du document courant. Un exemple d'heuristique pour l'extraction d'une seule composante est de prendre tous les exemples qui ne sont pas sur un chemin dont le mot de symboles[2] est aussi celui d'un exemple étiqueté positif.

Je termine cette section par une petite critique en forme de perspective. Dans ce cadre interactif, nous supposons aussi que lorsque l'algorithme passe de la composante $i$ à la composante $i + 1$, l'hypothèse de l'étape $i$ est supposée correcte et donc immuable. Les interactions suivantes ne porteront alors que sur des corrections pour la composante $i + 1$. Cette condition qui interdit de revenir sur l'apprentissage d'une composante de rang inférieur est très pénalisante. Elle devra être contournée.

### 3.2.4 Réalisations

Patrick a implanté PAF et l'a intégré à la plate-forme de génération de programmes d'extraction. Il l'a évalué sur un grand nombre de jeux de tests de deux natures. Le premier type de jeu de données a été produit par Fabien Torre à partir d'enregistrements d'une de nos bases de données. Ces données plutôt synthétiques ont été générées dans le

[2]Le mots de symboles d'un chemin est le mot composé de la concaténation de tous les symboles rencontrés sur ce chemin à partir de la racine.

but de reproduire toutes les organisations de données que nous avons identifiées, à grande échelle, afin d'éviter une tâche manuelle laborieuse d'annotation des corpus nécessaires à la validation. Le second type de jeux de données a été collecté sur Internet puis annoté.

Patrick a comparé les résultats de ces programmes à d'autres approches quand cela était possible (Beaucoup d'algorithmes ne sont pas disponibles publiquement ou ne produisent aucun résultat sur ces données). L'approche est convaincante pour moi car sans aucune connaissance du domaine ajoutée, sans traitement a posteriori particulier, en quelques interactions et quelques minutes on peut générer un programme d'extraction fini, capable directement d'alimenter une base de données. Sur de nombreux jeux de données reflétant la plupart des organisations identifiées lors de notre étude, les programmes d'extraction sont parfaits (précision et rappel à 100%).

## 3.3 Conclusion

Le travail de Patrick est d'abord un point de départ : pour confronter nos idées à des problèmes et des données réelles dans le cadre du projet Mostrare, pour prendre connaissance du problème à travers une meilleure analyse des données, leurs représentations, leurs codages et leurs organisations en `XML`. Il a pu ensuite introduire des méthodes d'apprentissage et de classification originales dans le processus d'extraction et développer un algorithme complet. À l'aide de la plate-forme commencée lors de la thèse de Julien, nous pourrons bientôt offrir sous forme de web service, le programme et les données à la communauté. D'un point de vue ingénierie, il sera aussi intéressant de fournir une sortie des programmes d'apprentissage dans un langage connu, lisible et éventuellement maintenable.

À mon sens la perspective principale de ce travail est d'intégrer des techniques d'apprentissage actif lors des interactions avec l'utilisateur. L'algorithme d'apprentissage doit suggérer de nouveaux exemples ou documents à étiqueter pour converger plus rapidement vers la solution.

Une deuxième perspective est plutôt une voie alternative et consiste à étudier comment éviter toute annotation de la part de l'utilisateur. Pour garder la même qualité dans la procédure d'extraction, il faut compenser l'absence de ces annotations par des connaissances a priori. Aujourd'hui beaucoup d'espoirs sont mis dans la définition et l'usage d'ontologies. Elles peuvent par exemple fournir une première annotation des documents approximative et bruitée ou incomplète. Dans cet esprit une collaboration a commencé avec Pierre Sénelart de l'équipe GEMO[3]. Nous devons adapter nos méthodes dans cette direction. Par ailleurs, le papier Abiteboul and Senellart (2006) propose déjà une forme probabiliste de mémorisation de données `XML` qui peut servir pour le stockage des résultats de l'application de procédures d'extraction non exactes.

Pour attaquer ce problème, je voudrais dans un premier temps en obtenir une définition plus formelle et étudier l'annotation de documents par des procédures stochastiques, probabilistes. C'est un sujet qui est abordé dans le chapitre suivant.

[3] `http://gemo.futurs.inria.fr/`

## 3.4 Notes bibliographiques

Le livre de Cornuéjols and Miclet (2002) est un ouvrage récent, pédagogique et complet sur l'apprentissage automatique (il est de surcroît en français). Plusieurs chapitres sont consacrés à la classification supervisée. Il est préfacé par Tom Mitchell, auteur aussi d'un livre sur l'apprentissage automatique qui reste aujourd'hui une référence (Mitchell, 1997). C'est aussi dans l'équipe de T. Mitchell que le projet WebKb[4] a été développé. Le projet a donné lieu à de nombreux travaux sur l'extraction et la recherche d'informations à l'aide de techniques d'apprentissage automatique. Les chercheurs qui y ont participé ont poursuivi ces efforts dans plusieurs groupes, comme par exemple D. Freitag et A. McCallum. J'invite le lecteur à consulter le site de WebKB et le résumé du projet Craven et al. (2000). Dans le même temps, l'équipe de C. Knoblock avec particulièrement le travail sur Stalker (Muslea et al., 1998) et le projet Ariadne (Knoblock et al., 2001) a aussi contribué à ce champ de recherches toujours très actif. Aujourd'hui, le nombre de publications sur l'extraction d'informations à partir de pages Internet est très impressionnant et forme un domaine particulier sur CiteSeer[5]. La communauté s'est aussi structurée autour de challenges et de référentiels de jeux de données comme RISE [6]. Elle a aussi pris conscience de la difficulté de l'évaluation de ces méthodes d'extraction dans Califf et al. (2004). Quelques articles de synthèse ont été publiés (voir (Kaiser and Miksch, 2005) pour un article récent).

Beaucoup de ces travaux concernent uniquement l'extraction d'information monadique et bien peu l'extraction $n$-aire. En revanche de nombreuses techniques de classification supervisée ou de fouille de données ont été utilisées : le boosting (Freitag and Kushmerick, 2000), des méthodes d'apprentissage actif (Muslea et al., 2002), des méthodes de maximum d'entropie (Chieu and Ng, 2002). Des modèles statistiques, comme les chaînes de Markov, sont aussi appliqués au problème d'extraction (Freitag and McCallum, 1999) mais nous reviendrons sur ces techniques dans le chapitre suivant.

Réduire le temps de définition d'un programme d'extraction et limiter le travail d'annotation de la part de l'utilisateur est une volonté que l'on retrouve affichée dans plusieurs groupes. Des systèmes interactifs (Ceresna, 2005) et des algorithmes non supervisés (Zhai and Liu, 2005; Papadakis et al., 2005; Liu et al., 2003; Crescenzi et al., 2001; Arasu and Garcia-Molina, 2003; Lerman et al., 2001, 2004) illustrent cette tendance. Toutefois, la sortie des programmes complètement non supervisés n'est pas directement exploitable comme peuvent l'être des algorithmes supervisés. Par exemple l'algorithme MDR de Liu et al. (2003) extrait souvent de larges parties de pages `HTML` si bien qu'un traitement a posteriori laborieux est nécessaire.

Le travail de Hurst (2001) est une approche complémentaire de l'étude que nous avons mené sur la présence de relations dans des documents arborescents. Il est concentré sur les tables et a pour origine les tables dans des documents textuels. Il a été exploité dans Lerman et al. (2001, 2004) pour construire des systèmes non supervisés. Cohen et al. (2002) extrait aussi des relations mais demande l'intervention complète de l'utilisateur pour la composition des enregistrements complets. Le travail le plus proche de notre contribution est certainement celui de Thomas (2003) avec une technique de programmation logique inductive.

[4] `http://www.cs.cmu.edu/~webkb`

[5] `http://citeseer.ist.psu.edu/InformationRetrieval/Extraction/`

[6] `http://www.isi.edu/info-agents/RISE/repository.html`

# Bibliographie

Abiteboul, S. and Senellart, P. (2006). Querying and Updating Probabilistic Information in XML. In *Extending DataBase Technology*, Munich, Germany.

Arasu, A. and Garcia-Molina, H. (2003). Extracting structured data from web pages. In *Proceedings of international conference on Management of data*, pages 337–348.

Califf, M. E., Ciravegna, F., Freitag, D., Giuliano, C., Kushmerick, N., Lavelli, A., and Romano, L. (2004). A critical survey of the methodology for ie evaluation. In *Proceedings of LREC 2004*.

Ceresna, M. (2005). Interactive generation of html wrappers using attribute classification. In *Proceedings of the First International Workshop on Representation and Analysis of Web Space*, pages 137–142, Prague-Tocna, Czech Republic.

Chieu, H. L. and Ng, H. T. (2002). A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of Eighteenth national conference on Artificial intelligence*, pages 786–791.

Cohen, W. W., Hurst, M., and Jensen, L. S. (2002). A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the eleventh international conference on World Wide Web*, pages 232–241. ACM Press.

Cornuéjols, A. and Miclet, L. (2002). *Apprentissage artificiel ; concepts et algorithmes*. Eyrolles.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigan, K., and Slattery, S. (2000). Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118(1–2) :69–113.

Crescenzi, V., Mecca, G., and Merialdo, P. (2001). Roadrunner : Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118.

Freitag, D. and Kushmerick, N. (2000). Boosted wrapper induction. In *AAAI/IAAI*, pages 577–583.

Freitag, D. and McCallum, A. K. (1999). Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*.

Hurst, M. (2001). Layout and language : Challenges for table understanding on the web. In *Proceedings of the 1st International Workshop on Web Document Analysis*, pages 27–30.

Kaiser, K. and Miksch, S. (2005). Information extraction. a survey. Technical Report Asgaard-TR-2005-6, Vienna University of Technology, Institute of Software Technology and Interactive Systems.

Knoblock, C. A., Minton, S., Ambite, J. L., Ashish, N., Muslea, I., Philpot, A., and Tejada, S. (2001). The ariadne approach to web-based information integration. *Int. J. Cooperative Inf. Syst.*, 10(1-2) :145–169.

Lerman, K., Getoor, L., Minton, S., and Knoblock, C. A. (2004). Using the structure of web sites for automatic segmentation of tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 119–130.

Lerman, K., Knoblock, C. A., and Minton, S. (2001). Automatic data extraction from lists and tables in web sources. In *Proceedings of Automatic Text Extraction and Mining workshop (ATEM-01), IJCAI-01.*

Liu, B., Grossman, R. L., and Zhai, Y. (2003). Mining data records in web pages. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–606.

Mitchell, T. (1997). *Machine Learning.* McGraw-Hill.

Muslea, I., Minton, S., and Knoblock, C. (1998). Stalker : Learning extraction rules for semistructured, web-based information sources. In *Procceeding of AAAI-98 workshop on AI and Information Extraction.*

Muslea, I., Minton, S., and Knoblock, C. (2002). Active + Semi-supervised Learning = Robust Multi-view Learning. In *Proceedings of ICML-2002*, pages 435–442.

Papadakis, N. K., Skoutas, D., Raftopoulos, K., and Varvarigou, T. A. (2005). Stavies : A system for information extraction from unknown web data sources through automatic web wrapper generation using clustering techniques. *IEEE Trans. Knowl. Data Eng.*, 17(12) :1638–1652.

Thomas, B. (2003). Bottom-up learning of logic programs for information extraction from hypertext documents. In Springer-Verlag, editor, *In proceedings of European Conference on Machine Learning / Principles and Practice of Knowledge Discovery in Databases ECML/PKDD 2003.*

Zhai, Y. and Liu, B. (2005). Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web*, pages 76–85.

# Chapitre 4

# Arbres et Champs aléatoires

Ce chapitre décrit le travail réalisé dans le cadre de la thèse de Florent Jousse. L'objectif est d'étudier les modèles stochastiques pour l'annotation d'arbres et de réaliser l'apprentissage automatique de ces modèles. La tâche d'annotation d'arbres est proche à la fois de l'extraction et de la transformation d'arbres. Nous avons proposé un modèle adapté aux arbres `XML` et défini des algorithmes d'apprentissage. Cette direction de recherche nous permet d'aller plus encore vers des méthodes approximatives avec l'objectif de traiter des problèmes où les données sont moins sûres et moins homogènes.

Les résultats sont obtenus en collaboration avec Rémi Gilleron, Isabelle Tellier et Florent Jousse.

## 4.1 Annotation d'arbres

L'annotation d'arbres consiste à décorer un arbre à l'aide d'étiquettes données. Dans le chapitre 2, les automates d'arbres ont servi de mécanisme pour réaliser une annotation d'arbres ensuite interprétée comme une extraction. En fait, c'est la sémantique donnée aux étiquettes (« extraire » et « ne pas extraire ») qui permet d'interpréter l'opération comme une extraction. En étendant l'alphabet de ces étiquettes et leur associant une sémantique particulière on peut facilement étendre la gamme de problèmes qui peuvent se réduire finalement à une opération d'annotation.

Certaines transformations d'arbres peuvent être vues sous la forme de problèmes d'annotation par au moins deux moyens. D'abord, la vue traditionnelle qui consiste à représenter les couples arbre en entrée, arbre transformé dans un unique arbre, par superposition (voir la figure 1.5). On se focalise alors sur l'étude de ces langages d'arbres (de superpositions). L'autre approche fait l'hypothèse que la transformation peut s'exprimer par des opérations réalisées à chaque noeud de l'arbre d'entrée, comme par exemple un homomorphisme, de la réécriture ou des opérations d'édition. On étiquette alors l'arbre d'entrée par les opérations réalisées à chaque noeud pour obtenir l'arbre de sortie.

Nous avons voulu aborder ces problèmes d'annotation avec des méthodes stochastiques à l'instar de ce qui est couramment fait dans le cas des séquences. Sa formulation est alors la suivante : un champ de variables aléatoires $\boldsymbol{X} = (X_1, \ldots, X_n)$ est observé, un champ $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$ de même taille correspond à l'annotation. Les valeurs observées et d'annotation forment les vecteurs $\boldsymbol{x} = (x_1, \ldots, x_n)$ et $\boldsymbol{y} = (y_1, \ldots, y_n)$.

| observation $\boldsymbol{x}$ | Mésange | charbonnière | ( | Parus | Major | ) |
|---|---|---|---|---|---|---|
| champ $\boldsymbol{X}$ | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ |
| annotation $\boldsymbol{y}$ | 0 | 0 | 0 | 1 | 1 | 0 |
| champ $\boldsymbol{Y}$ | $Y_1$ | $Y_2$ | $Y_3$ | $Y_4$ | $Y_5$ | $Y_6$ |

Ce qu'on cherche à modéliser, c'est la probabilité conditionnelle d'obtenir une annotation sachant l'observation $P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$. Ayant cette distribution en main, et étant donnée une observation $\boldsymbol{x}$, il suffit ensuite de calculer $\text{argmax}_{\boldsymbol{y}} P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$ pour connaître la meilleure annotation de cette observation.

Trois problèmes s'offrent à nous : comment modéliser cette distribution ? Comment apprendre cette distribution à partir d'exemples ? Comment calculer efficacement la meilleure annotation ?

Une proposition pour la modélisation de la distribution est de considérer un modèle génératif. C'est une proposition qui a longuement été reprise dans la littérature et souvent pour le cas des séquences. Si on sait calculer la probabilité de générer l'observation $P(\boldsymbol{X} = \boldsymbol{x})$ ainsi que la probabilité jointe de générer l'observation et l'annotation $P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})$ alors par la règle de Bayes on obtient facilement la probabilité conditionnelle :

$$P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x}) = \frac{P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})}{P(\boldsymbol{X} = \boldsymbol{x})}$$

Un modèle génératif pour les séquences ou les arbres est essentiellement un langage probabiliste de séquences ou d'arbres. Apprendre un tel langage, qui est notre second problème, est encore une question peu étudiée dans le cas des arbres. C'est un des objectifs du projet Marmota. Il semble toutefois qu'il soit nécessaire de disposer d'un grand ensemble de documents annotés pour estimer la distribution jointe $P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})$ et cette contrainte ne semble pas adaptée aux objectifs que nous nous fixons. En effet, nous tenons à limiter le travail de l'utilisateur final pour résoudre la tâche d'annotation.

Une seconde proposition est de modéliser directement la probabilité conditionnelle $P(\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{X} = \boldsymbol{x})$. On parle alors de modèles conditionnels. Nous avons suivi cette voie, guidés par le travail de travail récent de Lafferty et al. (2001) sur les champs conditionnels aléatoires.

## 4.2 Champs conditionnels aléatoires

Les champs conditionnels aléatoires ou `CRFs` sont un cadre pour définir des modèles probabilistes non génératifs, c'est-à-dire qui représentent une distribution de probabilité conditionnelle. Je vais d'abord introduire les `CRFs` à travers l'exemple de l'annotation de séquences.

Si on désire extraire les noms latins d'oiseaux de la liste en figure 4.1, on annotera chaque mot avec une information booléenne, notée ici 0 ou 1. Dans ce problème, on considère le texte comme une donnée observée.

La définition d'un `CRF` repose sur l'expression de fonctions qui traduisent la connaissance du domaine. Ces fonctions sont appelées par la suite des **features**. Les features sont des fonctions à valeurs réelles qui vont par exemple traduire le fait qu'un mot latin termine souvent par *a* ou par *i*, ou ne comporte jamais d'accent. Pour évaluer cette feature, il suffit de consulter le texte, l'observation. Par exemple, on pourrait traduire nos connaissances du latin par :


FIG. 4.1 – Liste des noms d'oiseaux en latin et français

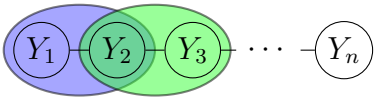



FIG. 4.2 – Graphe de dépendances dans le cas des séquences.

– $f_1(y_i, x_i) = 1$ si $y_i = 1$ et $x_i$ se termine par $a$ ou par $i$ ;
– $f_2(y_i, x_i) = 1$ si $y_i = 0$ et $x_i$ contient *é*.

On pourrait bâtir un modèle à l'aide d'un ensemble de telles features et d'un jeu de paramètres à valeurs réelles qui pondèrent leur poids. Par exemple attribuer un poids assez élevé à la feature $f_2$ est sans doute intéressant puisque les mots latins ne contiennent jamais de *é*.

Mais la connaissance apportée simplement par la lecture de l'observation peut se révéler insuffisante dans bien des cas. Il existe des noms français d'oiseau se terminant par $i$ comme la bourscarle de Cetti. Pour contourner cette difficulté, on peut ajouter une feature qui indique qu'un mot est sans doute un nom latin si son voisin l'est aussi. Cette feature exprime une dépendance entre deux annotations voisines. L'annotation d'un mot n'est plus simplement conditionnée par l'observation mais dépend aussi de l'annotation d'autres mots. Les features vont alors s'écrire selon la forme :

– $f_3(y_{i-1}, y_i, x_i) = 1$ si $y_i = 1$ et $y_{i-1} = 1$ et $x_i$ se termine par $a$ ou par $i$ ;

La dépendance entre les annotations est traduite par la forme et la définition sémantique des features qui traduisent ces connaissances du domaine. On notera que l'expression de ces dépendances dans les features est symétrique et non dirigée : $y_i$ dépend de $y_{i+1}$ et vice versa. Le modèle sera alors plus expressif mais on notera aussi que le calcul de l'annotation la plus probable sera aussi plus complexe. Dans notre exemple, il faudra considérer l'annotation de couples en couples de mots sur la longueur du texte. Limiter les dépendances est essentiel pour préserver un critère praticable pour les algorithmes qui manipuleront le modèle.

D'un point de vue formel, on va considérer un champ aléatoire $\boldsymbol{X}$ pour les observations et un champ $\boldsymbol{Y}$ pour leur annotation. Un `CRF` va représenter la distribution conditionnelle $P(\boldsymbol{Y}|\boldsymbol{X})$.

Les relations de dépendances et d'indépendance entre les variables dans le champ $Y$ sont exprimées par un graphe dont les noeuds sont associés aux $Y_i$. On n'exprime pas de dépendance avec les variables observées car elles sont données. C'est là une différence essentielle avec les modèles génératifs car on calculera une probabilité étant donnée une observation. Une conséquence importante est que chaque feature pourra poser des conditions sur toute l'observation $\boldsymbol{X}$ et non seulement sur la variable $X_i$ à la position courante du champ de variables. Chaque arc dans le graphe est non dirigé et exprime une dépendance. L'absence d'arc signale une indépendance. Le graphe définit un **voisinage** pour chaque variable dont la valeur ne dépendra que de la valeur de ses voisines. Dans le cas des séquences, on fixe souvent le graphe selon cette forme, exploitant l'ordre de succession des lettres : chaque $Y_i$ est connecté à son successeur et son prédécesseur. Cela donne le graphe de la figure 4.2. Choisir ce graphe a priori limite la forme des features et donc des connaissances que l'on injecte dans le modèle.

Étant donné ce graphe, on ne pourra formuler des features que de la forme $f(y_i, \boldsymbol{x}, i)$ ou $f(y_{i-1}, y_i, \boldsymbol{x}, i)$. L'expression $f(y_{i-1}, y_i, \boldsymbol{x}, i)$ désigne une feature qui reçoit : la valeur du champ $\boldsymbol{Y}$ aux points $i-1$ et $i$, soient deux valuations $y_{i-1}$ et $y_i$ ; ainsi que toutes les

observations de $\boldsymbol{x}$ nécessaires pour effectuer son calcul au point $i$, noté par commodité $\boldsymbol{x}, i$. Pour plus de simplicité, par la suite on rassemble toutes les features (de tout type) sous une forme unique indicée par $k$ : $f_k(y_{i-1}, y_i, \boldsymbol{x}, i)$.

Le modèle de probabilités ainsi défini par son graphe $G$ tombe dans la classe des modèles graphiques non dirigés. Grâce à un résultat fondamental des années 70, le théorème d'Hammersley-Clifford, on sait que la distribution peut s'exprimer par un produit de fonctions locales à chaque clique[1] du graphe :

$$P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{c \in C_G} \Phi_c(\boldsymbol{y}_c, \boldsymbol{x}) \ .$$

Ici, on note $C_G$ l'ensemble des cliques de $G$ et $\boldsymbol{y}_c$ le champ $y$ restreint à une clique $c$. Les $\Phi_c$ sont des fonctions réelles et $Z(\boldsymbol{x})$ est un coefficient de normalisation.

La définition des `CRFs` va faire en sorte que la paramétrisation de cette distribution sera log-linéaire en les features donc que chaque $\Phi_c$ soit en fait l'exponentielle de la somme pondérée des features qui s'appliquent dans la clique $c$ :

$$\Phi_c(\boldsymbol{y}_c, \boldsymbol{x}) = \exp\big(\sum_k \lambda_k f_k(\boldsymbol{y}_c, \boldsymbol{x}, c)\big) \ .$$

On obtient la classe des distributions représentées par un `CRF` sur les séquences, pour le graphe de dépendances de la figure 4.2 :

$$P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\left(\sum_{i=1}^{n} \sum_k \lambda_k f_k(y_{i-1}, y_i, \boldsymbol{x}, i)\right) \ . \tag{4.1}$$

Le choix d'une telle paramétrisation est motivé par les propriétés analytiques de ces fonctions. On retrouvera de bonnes dispositions pour ajuster les paramètres selon le principe de maximum de vraisemblance : étant donné un échantillon étiqueté, trouver le jeu de paramètres qui maximise la probabilité d'avoir généré cet échantillon.

Il reste à préciser que la définition a été donnée pour un champ aléatoire de taille fixée et un graphe de dépendances fixé. Dans la réalité, on appliquera ce modèle à toute séquence. On considère alors que chaque définition de feature $f_k$ s'applique indifféremment en tout point de la séquence et que son paramètre associé $\lambda_k$ est identique en chacun de ces points. C'est ce qui est déjà annoncé par l'équation (4.1) où le $k$ n'est pas indicé par $i$.

Trois problèmes algorithmiques principaux se posent dans ce chapitre

1. Le calcul de la probabilité d'une annotation.
2. Le calcul de la meilleure annotation d'une observation : l'**inférence**.
3. L'ajustement des paramètres d'un `CRF` en fonction d'un échantillon : l'**entraînement**.

Tous donnent un algorithme non praticable s'ils sont pensés un peu naïvement, puisqu'il est nécessaire de considérer toutes les annotations possibles d'une séquence $\boldsymbol{x}$ au moins dans le calcul du coefficient de normalisation $Z(\boldsymbol{x})$. Dans le cas des séquences, avec le modèle de dépendances de la figure 4.2, on utilise des algorithmes de programmation

[1] Parties du graphe $G$ totalement connectées.

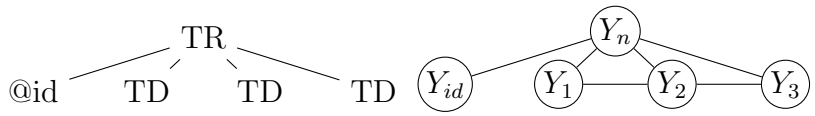



FIG. 4.3 – Modèle de dépendances dans les arbres XML.

dynamique très proches de Viterbi, bien connus dans les HMMs. Le troisième problème est plus complexe. Pour l'ajustement des paramètres, basé sur la maximisation de la vraisemblance, on exploite la forme de la fonction (4.1). La forme log-linéaire garantit que la fonction admet un maximum global unique. Il suffit donc de calculer le point où s'annule la dérivée. Mais on ne peut obtenir analytiquement une solution générale à ce problème. On utilise donc des algorithmes de descente de gradient pour approcher ce point.

## 4.3 Adaptation aux arbres

Nous voulons adapter le modèle proposé sur les séquences aux arbres XML. Les tâches d'annotation, extraction ou transformation demandent d'observer et de manipuler autant les noeuds éléments, que les noeuds texte ou attribut d'un arbre DOM. On peut aussi être amenés à traiter des parties de feuilles texte.

Cette adaptation suppose une réflexion sur la nature des features que nous pouvons construire et donc la forme du modèle de dépendances. Le choix de ce modèle a ensuite une incidence sur les algorithmes qui permettent d'entraîner les CRFs, de calculer la probabilité d'une annotation ou de rechercher la meilleure annotation. Nous avons appelé ce nouveau modèle les XCRFs.

### 4.3.1 Modèle de dépendances

Dans les séquences, le voisinage dans le graphe de dépendances suit la relation d'ordre sur les éléments de la séquence. Les arbres XML sont une structure soumise à deux ordres : l'ordre père-fils et l'ordre frère-frère suivant. Dans la définition des types de documents XML par des schémas ou des DTD, l'ordre frère-frère suivant est conditionné par l'étiquette du père.

Nous avons voulu traduire la possibilité de conditionner l'annotation des arbres en exploitant ces deux ordres naturels dans les arbres XML. Dans les features des XCRFs, on aura donc accès à trois étiquettes de l'annotation qui sont un père et deux de ses fils consécutifs, en ce qui concerne les noeuds élément et les noeuds texte. En ce qui concerne les attributs, la notion d'ordre entre les fils n'est pas pertinente. Seul l'ordre père fils subsiste. Le modèle de dépendances est donc celui illustré dans la figure 4.3.

D'un point de vue formel, la définition de la probabilité n'est pas très différente du cas des séquences. Les features ont trois paramètres portant sur les annotations : $y_n$ pour le père et $y_{n.i}, y_{n.(i+1)}$ pour deux fils consécutifs. Le couple $\boldsymbol{x}, n.i$ est une notation pour désigner toute l'information de $\boldsymbol{x}$ nécessaire pour évaluer la feature à une position donnée dans l'arbre.

$$P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \exp\left( \sum_{i=1}^{n} \sum_{k} \lambda_k f_k(y_n, y_{n.i}, \boldsymbol{y}_{n.(i+1)}, \boldsymbol{x}, n.i) \right) , \qquad (4.2)$$

avec le coefficient de normalisation $Z(\boldsymbol{x})$ défini par :

$$Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \exp\left( \sum_{i=1}^{n} \sum_{k} \lambda_k f_k(y_n, y_{n.i}, \boldsymbol{y}_{n.(i+1)}, \boldsymbol{x}, n.i) \right) . \tag{4.3}$$

### 4.3.2 Expressivité

Le choix du modèle de dépendances est un compromis entre expressivité et complexité. La complexité est fortement dépendante de la taille de la plus grande clique. Nous obtenons des cliques de taille 3 dans les `XCRF` qui nous donneront des algorithmes intégrant un facteur cubique dans le nombre d'étiquettes possibles. Une alternative conservant des cliques de taille 2 aurait par exemple été de ne pas tenir compte de l'ordre selon les frères ou de voir la donnée `XML` comme un simple texte déjà découpé en unités lexicales grâce aux balises.

L'intérêt de choisir ce modèle de dépendances est qu'il nous permet de représenter une classe de distributions plus grande qu'avec des dépendances plus réduites. Nous avons montré que toute distribution conditionnelle représentée par des automates d'arbres stochastiques était aussi représentable par un `XCRF`.

Les automates d'arbres stochastiques sont vus ici comme des modèles génératifs qui permettent d'étiqueter un arbre avec les états de l'automate. Nous obtenons une distribution de probabilités conditionnelle en suivant la règle de Bayes comme indiqué dans la section 4.1. La preuve est directe. Un automate stochastique étant donné par un ensemble de règles pondérées, il s'agit de traduire ces règles en features. Les poids des règles se traduisent en poids des features au log près.

Nous dépassons toutefois l'expressivité des automates pour deux raisons. D'une part nous pouvons annoter les parties non ordonnées comme les attributs. D'autre part, la modélisation directe d'une probabilité conditionnelle nous permet de nous affranchir de la modélisation de l'observation et donc nous pouvons intégrer des connaissances « à longue distance » dans l'observation.

L'intérêt de cette correspondance est aussi plus pratique. Nous avons travaillé sur l'inférence d'automates d'arbres. L'inconvénient était une certaine rigidité due à l'approche purement syntaxique de l'inférence grammaticale. Se tourner vers l'inférence d'arbres stochastique me semble difficile dans le cadre de l'annotation interactive car elle demande en général des grands ensembles d'arbres annotés pour l'entraînement. Une idée est alors de contourner cette rigidité en transformant l'automate en `CRF`. L'autre façon d'interpréter cette idée est de voir l'inférence grammaticale comme une façon de sélectionner des features pour un modèle non génératif comme les `CRFs`.

### 4.3.3 Inférence et entraînement

Le calcul de la probabilité d'une annotation demande de calculer le coefficient de normalisation $Z(\boldsymbol{x})$ défini par l'équation (4.3). Ce calcul effectué naïvement donne un algorithme impraticable car on doit considérer toutes les annotations possibles de $\boldsymbol{x}$. Comme dans le cas des séquences, il est possible d'utiliser un algorithme de programmation dynamique pour réaliser le calcul, les relations d'indépendance permettant de mémoriser des calculs intermédiaires et donc de faire chuter la complexité. La différence essentielle qui induit une difficulté supplémentaire dans notre cas, est la présence des

deux ordres qui nécessitent une double récursion en largeur et en hauteur dans les arbres XML.

En un certain sens, l'algorithme obtenu s'inspire à la fois de l'algorithme d'inférence dans les HMMs pour les fils d'un noeud et de l'algorithme d'inférence dans les grammaires algébriques probabilistes pour les relations père-fils.

La complexité du calcul est linéaire dans la taille de l'arbre mais cubique dans la taille de l'alphabet d'annotation. Mais il faut intégrer deux remarques. Premièrement, c'est une complexité dans le pire des cas mais qui est toujours atteinte. Autrement dit c'est aussi une borne minimale pour cet algorithme. Deuxièmement, la complexité n'intègre pas le calcul des features. Il convient donc à la fois de choisir des ensembles d'étiquettes de taille raisonnable et des features dont le calcul n'est pas rédhibitoire.

Comme indiqué dans la section 4.2, le vecteur des paramètres $\lambda_k$ qui va maximiser la vraisemblance d'un échantillon ne peut pas être déterminé analytiquement. L'entraînement d'un XCRF consiste alors à approcher ce vecteur optimal. La méthode utilisée est une descente de gradient qui demande de calculer la dérivée de la vraisemblance par rapport à chaque $\lambda_k$. Une nouvelle fois ce calcul, pour être efficace, met en oeuvre deux récursions et la mémorisation de résultats intermédiaires. L'algorithme de programmation dynamique obtenu utilise des coefficients qui sont à la fois à l'image des coefficients inside/forward des HMMs et forward/backward dans les grammaires algébriques probabilistes. La complexité reste identique à l'inférence pour un pas de gradient.

### 4.3.4 Réalisations

Le système XCRF a été implanté par Missi Tran. Il est disponible à partir de la forge de l'INRIA à l'adresse http://treecrf.gforge.inria.fr. La description d'un XCRF est faite par un fichier XML et les fonctions features s'écrivent à l'aide d'expressions XPath. Le système permet d'annoter des arbres ou des textes.

Des expériences ont été menées principalement sur deux jeux de données. L'un est issu d'un challenge Pascal dont les données ont été préparées par Denoyer et al. (2006) à partir d'un travail que j'avais réalisé sur de multiples transformations dans des formats XML ou HTML de la base de données International Movie Database. Il a servi à faire une tâche d'annotation et une tâche de transformation. La transformation est représentée par des opérations d'édition simples comme « remplacer un symbole » ou « supprimer un noeud ». Le second est disponible sur le site de A. Doan. Il a servi pour réaliser une tâche de schéma matching : annoter des documents provenant de sources différentes (typés selon des schémas XML différents) avec les éléments d'un schéma médiateur unique.

Les expériences ont montré l'intérêt des features sur les cliques à trois noeuds dans des cas pratiques. Les temps d'inférence et d'entraînement sont raisonnables. Elles ont aussi montré qu'on pouvait atteindre des performances acceptables pour ces tâches assez différentes. Nous dépassons par exemple les performances de Doan avec son système LSD sans utiliser de connaissance du domaine.

## 4.4 Conclusion

Adapter les CRF à l'annotation d'arbres et particulièrement aux documents XML était pour le projet Mostrare un pas vers les techniques d'apprentissage statistique. Elles

ont permis d'attaquer des sources de données plus hétérogènes et devraient être plus résistantes à quelques bruits de structure ou de contenu. Nous avons montré que notre adaptation était pertinente et qu'elle était rapide et performante.

Le travail doit se poursuivre dans plusieurs directions. Pour limiter la complexité, il faut éviter de considérer des ensembles d'étiquettes trop grands. Une solution élégante est itérative et passe par des abstractions successives de l'étiquette finale par des ensembles d'étiquettes. Dans un premier temps, les étiquettes sont rassemblées en groupes et un premier `CRF` doit annoter les arbres par l'identifiant de ces groupes. Un second `CRF` prend cette première annotation comme une observation et raffine un groupe en groupes plus petits. Cette imbrication hiérarchique d'application de `CRFs` s'opère selon une relation hiérarchique des étiquettes. Cette hiérarchie peut être donnée, selon les tâches, par une ontologie pour une tâche d'annotation simple ou par le type des documents produits pour une transformation. C'est une idée importante car elle constitue un premier pas pour l'utilisation d'ontologies pour améliorer ou aider les tâches d'extraction ou de transformation.

Une seconde direction propose de combiner l'annotation des noeuds de l'arbre avec l'annotation des textes dans ses feuilles. Il faut dans un premier temps opérer un découpage en unités lexicales des feuilles. Plusieurs solutions sont en cours d'élaboration consistant à représenter le texte par un arbre filiforme ou à simplement introduire autant de noeuds texte qu'il y a d'unités lexicales.

En ce qui concerne les transformations encore très peu abordées, une question essentielle est de pouvoir obtenir des annotations à partir des couples de documents (entrée,sortie). Ce travail suppose de mieux étudier les distances d'édition entre arbres et les transformations, un sujet qui sera traité dans le projet Marmota.

## 4.5 Notes bibliographiques

Les `XCRFs` font partie de la famille des modèles graphiques. Mariage entre théorie des graphes et des probabilités, les modèles graphiques regroupent les définitions de distributions de probabilités accompagnées de notions de modularité exprimées par présence de graphes qui traduisent des relations d'indépendance entre variables aléatoires. Cette modularité a des conséquences importantes en terme de complexité algorithmique. De ce fait les modèles graphiques sont aussi un moyen de ramener les probabilités dans le champ des sciences pour l'ingénieur. Ce domaine fait l'objet d'une production scientifique soutenue depuis longtemps.

Certains modèles génératifs comme les `HMMs`, les automates stochastiques (sur les séquences ou les arbres), et les `PCFGs` entrent dans cette catégorie. Ces modèles génératifs représentent une distribution jointe et dont le graphe de dépendances est dirigé (dépendance de l'observation à l'état courant et de l'état courant à l'état précédent dans le cas des `HMMs`). Les `HMMs` ont particulièrement été appliqués sur des séquences. Bréhélin and Gascuel (2000) est un bon article d'introduction sur les `HMMs`. Les automates de mots stochastiques représentent aussi des distributions de probabilité (jointes) sur des séquences. Syntaxiquement, leur définition repose sur la donnée d'un automate, équipé de transitions pondérées. Les poids des transitions sortant d'un même état somment à 1 pour définir une mesure de probabilité sur les séquences. Automates stochastiques et `HMMs` représentent en fait les mêmes distributions (Dupont et al., 2005). Mais le point

de vue de la théorie des langages apporte des éclaircissements sur la nature de ces distributions, à travers des formalismes plus généraux comme les automates pondérés, les automates à multiplicités, les séries rationnelles. En témoignent les travaux en apprentissage automatique récents de Denis et al. (2006).

Les PCFGs sont des modèles probabilistes pour l'analyse dans les grammaires algébriques. Syntaxiquement, les PCFGs sont des grammaires algébriques équipées de poids sur les transitions. Elles s'appliquent alors sur des arbres d'analyse. L'ensemble des arbres de dérivation de toute grammaire algébrique forme un ensemble régulier d'arbres, donc on peut voir une PCFGs comme un modèle représentant une distribution de probabilités pour des langages d'arbres réguliers. Les applications qui ont contribué à leur expansion sont d'abord dans le champ du traitement automatique des langues (Manning and Schütze, 1999). Des questions importantes restent toutefois ouvertes dans ce domaine puisqu'on ne dispose que d'un semi algorithme pour décider si une grammaire algébrique équipée de poids représente une distribution de probabilités (Wetherell, 1980). De plus, tous les langages réguliers ne sont pas des ensembles d'arbres d'analyse dans une grammaire. Il me semble que dans le cas des documents structurés comme XML l'approche par des outils plus généraux que les grammaires algébriques, par exemple les automates d'arbres ou les séries rationnelles d'arbres, apporterait une meilleure compréhension de ces formalismes ou de leurs algorithmes. C'est une étude en cours dans le projet Marmota.

Mais dans la littérature, les modèles graphiques considérés sont d'abord des réseaux bayésiens (Jordan, 1998; Lauritzen, 1996). C'est avec ce modèle de distribution comme cible (ou simplement des champs de Markov) que les algorithmes principaux ont été définis : Belief propagation, ou Sum-Product (Min-Sum), Junction-Tree algorithm (Hugin ou Shafer-Shenoy). Ils s'appliquent toutefois à tous les modèles graphiques. L'algorithme de Viterbi d'inférence dans les HMMs, plus ancien, peut être vu comme une instance de ces algorithmes généraux, tout comme celui de l'inférence dans les PCFGs et celui que nous avons proposé dans ce chapitre.

En ce qui concerne l'entraînement, c'est-à-dire la recherche des paramètres du modèle, les algorithmes reposent sur différentes techniques selon la nature analytique des fonctions qui régissent ces distributions et les structures qu'elles modélisent. Ce sont souvent des procédures qui maximisent la vraisemblance dans des itérations à la EM (Dempster et al., 1977). La difficulté algorithmique réside dans une potentielle explosion combinatoire réglée par une méthode de programmation dynamique. L'algorithme des XCRFs de ce chapitre s'inspire, à la fois de l'algorithme inside-outside de Lari and Young (1990) pour les PCFGs et du forward-backward pour les HMMs. Il est aussi très proche de l'inférence dans les HMMs hiérarchiques de Fine et al. (1998).

Les CRFs s'inspirent donc à la fois des modèles graphiques et des méthodes de maximum de vraisemblance (ou encore de maximum d'entropie). Le théorème qui fonde les CRFs a été à la fois montré par Hammerlsey-Clifford et Besag dans les années 70. Il montre que les champs de Markov aléatoires définissent la même classe de distributions que les distributions de Gibbs et s'expriment comme une produit (normalisé) de fonctions de potentiel appliquées sur chaque clique du graphe. L'apport principal de Lafferty et al. (2001) a été de proposer une forme particulière de ces fonctions de potentiel, une forme exponentielle définie par des fonctions features. Cette forme sera alors adaptée à l'inférence et l'entraînement par maximum de vraisemblance. L'article de synthèse de Sutton and McCallum (2006) constitue une excellente introduction sur le sujet. Même si la forme de la (log-) vraisemblance d'un échantillon tiré selon une distribution repré-

sentée par un CRF a la bonne propriété d'avoir un maximum global unique, on ne peut le trouver de façon analytique. Il est nécessaire d'appliquer des méthodes de montée de gradient. Les travaux de Wallach (2002) et Malouf (2002) ont montré que celles initialement proposées (IIS) dans Lafferty et al. (2001) étaient beaucoup moins adaptées aux CRFs que celle de Beson and Moré (2001) ou celle (L-BFGS) de Byrd et al. (1995) que nous avons donc utilisé dans les XCRFs. Malgré cela, une critique des CRFs en général reste le temps de convergence dans l'entraînement. Plusieurs techniques pour réduire ce problème ont été proposées : la sélection de features, l'approximation, l'utilisation d'autres critères d'entraînement (Bayesian CRFs, noyaux).

La sélection de features dans les CRFs n'a pas donné pour le moment de résultats intéressants car les performances ne semblent pas au rendez-vous (McCallum, 2003).

La complexité des algorithmes est quadratique dans le nombre d'étiquettes (cubique dans le cas des XCRFs). Réduire le nombre d'étiquettes participe donc grandement dans une réduction des temps de calcul. Les CRFs dynamiques de Sutton et al. (2004) initialement proposés pour estimer la prédiction conjointe de plusieurs étiquettes dépendantes (*e.g.* entité nommées, catégories syntaxiques) sont aussi un moyen d'accélérer la convergence. L'idée est d'itérer la construction de CRFs, les étiquettes en sortie du $i^{\text{e}}$ servant de symboles observés pour le $i + 1^{\text{e}}$.

Une approche alternative et originale à la maximisation de la vraisemblance est proposée dans Taskar (2004). Elle consiste à intégrer dans le modèle des champs de Markov une approximation des paramètres selon le critère de maximisation des marges. C'est donc en quelque sorte, une approche noyau qui est utilisée au sein des champs de Markov dans un nouveau modèle M3N (Maximum Margin Markov Networks). Le récent travail de Spengler (2005) a montré comment les appliquer aux documents XML. Les M3Ns dépassent (en termes de qualité de prédiction) les CRFs, mais nous avons constaté que leurs performances étaient semblables aux XCRF. Une autre approche reposant sur les noyaux est proposée dans Lafferty et al. (2004) qui permet aussi une approche semi-supervisée de l'entraînement.

L'annotation de séquences est l'une des premières tâches à laquelle on a appliqué les CRFs, dès l'article qui les a introduits. Pinto et al. (2003) les ont ensuite appliqués à l'extraction dans les tables puis plus généralement dans le domaine du traitement statistique des textes (McCallum and Li, 2003; Cohn and Blunsom, 2005). Les motivation initiales de la thèse de Florent étaient d'ailleurs la construction automatique de motifs d'arbres à partir d'arbres syntaxiques dans les systèmes question-réponse (Jousse et al., 2005). Des systèmes d'extraction et d'annotation de texte comme Minorthird de Cohen[2] les utilisent. On peut aussi citer le projet de Sunita Sarawagi sur l'intégration de données (Mansuri and Sarawagi, 2006) et son implantation des CRFs [3]. Par ailleurs, Sarawagi and Cohen (2004) dépassent un peu la contrainte de dépendance markovienne avec les semi-markov CRF pour réaliser les tâches de segmentation en unités lexicales et d'extraction simultanément.

Un objectif de notre travail est de produire une sortie structurée sous forme d'arbres à partir de données elles aussi structurées. C'est aussi un des objectifs des travaux de l'équipe de P. Gallinari au LIP6. Un résultat particulièrement proche est celui de Gallinari et al. (2005), faisant suite à des études de la classification de document arbores-

[2] minorthird.sourceforge.net

[3] crf.sourceforge.net

cents (Denoyer and Gallinari, 2004). La considération de d'espaces de sortie complexes (des arbres par exemple) pour des tâches comme la classification est aussi le sujet de recherches actives dans la communauté des machines à vecteurs de support (Tsochantaridis et al., 2005). Enfin, l'annotation avec des opérations d'édition d'arbres est aussi un angle d'attaque de ce problème. Les récents résultats de l'équipe de St Etienne (participant à Marmota) nous aideront sans doute à mieux comprendre ce que l'on peut attendre d'un cadre stochastique (Bernard et al., 2006; Oncina and Sebban, 2005).

# Bibliographie

Bernard, M., Habrard, A., and Sebban, M. (2006). Learning stochastic tree edit distance. In *Proceedings of the 17th European Conference on Machine Learning (ECML06)*.

Beson, S. J. and Moré, J. J. (2001). A limited memory variable metric method for bound constrained optimisation. Technical Report ANL/ACS-P909-0901, Argonne National Laboratory.

Bréhélin, L. and Gascuel, O. (2000). *Le temps, l'espace et l'évolutif - Ecole thématique Document et Evolution*, chapter Modèles de Markov cachés et apprentissage de séquences. Cepadues-editions edition.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Computing*, 16(5) :1190–1208.

Cohn, T. and Blunsom, P. (2005). Semantic role labelling with tree conditional random fields. In *CoNLL'05 : Proceedings of The Ninth Conference on Natural Language Learning*.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society-B*, 39 :1–38.

Denis, F., Esposito, Y., and Habrard, A. (2006). Learning rational stochastic languages. In *Proceedings of the 19th Annual Conference on Learning Theory*, pages 274–288.

Denoyer, L. and Gallinari, P. (2004). Bayesian network model for semi-structured document classification. *Inf. Process. Manage.*, 40(5) :807–827.

Denoyer, L., Gallinari, P., and Vercoustre, A. M. (2006). Xml mining challenge at inex 2005. Technical report, University of Paris VI, INRIA.

Dupont, P., Denis, F., and Esposito, Y. (2005). Links between probabilistic automata and hidden markov models : probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9) :1349–1371.

Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden markov model : Analysis and applications. *Machine Learning*, 32(1) :41–62.

Gallinari, P., Wisniewski, G., and Denoyer, L. (2005). Stochastic models for document restructuration. In *In ECML Workshop on Relationnal Machine Learning*.

Jordan, M. I., editor (1998). *Learning in Graphical Models*. MIT Press.

Jousse, F., Tellier, I., Tommasi, M., and Marty, P. (2005). Learning to extract answers in question answering : Experimental studies. In *Actes de CORIA'05*, pages p.85–100. Hermès.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pages 282–289.

Lafferty, J. D., Zhu, X., and Liu, Y. (2004). Kernel conditional random fields : representation and clique selection. In *Proceedings of the Twenty-first International Conference*.

Lari, K. and Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4 :35 – 56.

Lauritzen, S. L. (1996). *Graphical Models*. Oxford University Press.

Malouf, R. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)*, pages 49–55.

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge.

Mansuri, I. and Sarawagi, S. (2006). A system for integrating unstructured data into relational databases. In *Proc. of the 22nd IEEE Int'l Conference on Data Engineering (ICDE)*.

McCallum, A. (2003). Efficiently inducing features of conditional random fields. In *Proceedings of the 19th Conference in Uncertainty in Artificial Intelligence*, pages 403–410.

McCallum, A. and Li, W. (2003). Early results for named entity recognition with conditional random fields. In *CoNLL'2003 : Proceedings of The Seventh Conference on Natural Language Learning*.

Oncina, J. and Sebban, M. (2005). Learning unbiased stochastic edit distance in the form of a memoryless finite-state transducer. In *Workshop on Grammatical Inference Applications : Successes and Future Challenges*.

Pinto, D., McCallum, A., Wei, X., and Croft, W. B. (2003). Table extraction using conditional random fields. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 235–242.

Sarawagi, S. and Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. In *Proceedings of NIPS*, pages 1185–1192.

Spengler, A. (2005). Maximum margin Markov networks for XML tag relabelling. Master's thesis, University of Karlsruhe (TH), Germany.

Sutton, C. and McCallum, A. (2006). *Introduction to Statistical Relational Learning*, chapter An Introduction to Conditional Random Fields for Relational Learning. MIT Press, lise getoor and ben taskar edition.

Sutton, C., Rohanimanesh, K., and McCallum, A. (2004). Dynamic conditional random fields : Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, pages 783–790.

Taskar, B. (2004). *Learning Structured Prediction Models : A Large Margin Approach*. PhD thesis, Stanford University.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6 :1453–1484.

Wallach, H. (2002). Efficient training of conditional random fields. Master's thesis, University of Edinburgh.

Wetherell, C. S. (1980). Probabilistic languages : A review and some open questions. *ACM Comput. Surv.*, 12(4) :361–379.

# Chapitre 5

# Arbres ou apprentissage

Dans les chapitres précédents, j'ai voulu montrer le travail accompli au sein du projet Mostrare. Mostrare est un projet récent (2003) et avant sa création j'ai pu poursuivre des recherches dans deux directions principales : les arbres[1], d'un point de vue théorique le plus souvent et l'apprentissage automatique.

## 5.1 Dans les arbres

### 5.1.1 Tata

Tata (Comon et al., 1997) est un livre sur les automates d'arbres librement disponible sur internet. Il a été écrit avec Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison et corrigé grâce à de nombreuses remarques de la part de relecteurs. L'idée de ce livre est de fournir une vue plus algorithmique sur les automates d'arbres et en accompagnant leur présentation d'exemples d'applications. Les premiers chapitres sont plutôt destinés à un public d'étudiants et enseignants chercheurs éventuellement d'autres disciplines.

Le livre a servi de support pour plusieurs cours en France ou à l'étranger et il est très fréquemment téléchargé. Référencé sur de nombreux sites, il est cité par plusieurs auteurs au sein du W3C. C'est un peu le premier regret que j'ai puisque qu'aucune application à `XML` n'apparaît et que le traitement des arbres d'arité non bornée n'est évoqué que très rapidement. Le temps et plus sûrement le courage m'ont manqué pour rédiger un nouveau chapitre sur ce sujet à la suite de nos travaux.

### 5.1.2 Réécriture, morphismes d'arbres et régularité

Une série de mes travaux un peu plus anciens concerne des questions de décision sur la régularité d'ensembles d'arbres. Dans les langages de mots, l'image d'un langage par morphisme est régulier. Mais dans le cas des arbres, cette propriété est perdue. Un morphisme $\tilde{h}$ d'arbres est une transformation définie par une fonction que l'on note $h$ sur les lettres de l'alphabet. La fonction $h$ transforme les lettres en des termes et s'étend en un morphisme qui réalise une transformation d'arbres.

[1] Dans tout ce chapitre, les arbres sont en fait des termes, l'arité est fixe.

Par exemple, considérons les deux alphabets $\Sigma_1 = \{f(), a\}$ et $\Sigma_2 = \{g(,), b\}$. La lettre $g$ est d'arité 2, $f$ est d'arité 1 alors que $a$ et $b$ sont des constantes qui apparaissent aux feuilles. On peut définir un morphisme d'arbres depuis les arbres écrits avec $\Sigma_1$ vers les arbres écrits avec $\Sigma_2$ par $h(f(x)) = g(x, x)$ et $h(a) = b$. La présence des variables est ici nécessaire à cause de l'arité des lettres, pour représenter par exemple le terme qui peut se placer sous un $f$ ou les termes qui peuvent se mettre en fils droit et gauche d'un $g$. Pour calculer l'image d'un arbre on applique $h$ récursivement à partir de la racine. Par exemple $f(f(f(a)))$ est transformé en $g(g(g(b, b), g(b, b)), g(g(b, b), g(b, b)))$. Dans mon exemple, $f$ est transformé en un terme $g(x, x)$ qui est dit non linéaire à cause de la répétition de la variable $x$. Cette non-linéarité entraîne une duplication de termes dans l'application du morphisme et c'est la raison principale de la perte de la propriété de régularité des images de langages par morphisme. Par exemple, l'image du langage régulier $\{f^n(a) \mid n \in N\}$ par $h$ n'est pas régulier (toutes les branches sont de même longueur, on ne peut vérifier cette propriété dans les langages d'arbres réguliers).

Le premier résultat, montré avec Max Dauchet et Sophie Tison (Dauchet et al., 2002), prouve que si l'image d'un langage régulier $R$ par un morphisme reste un langage régulier, alors cette image peut aussi être obtenue par un morphisme linéaire sur un sous ensemble de $R$. Il exprime en quelques sortes la façon d'éviter la non-linéarité.

La perte de la régularité entraîne souvent la perte de la décidabilité de problèmes importants. Dans un autre travail réalisé avec Franck Seynhaeve et Sophie Tison (Seynhaeve et al., 1999), nous avons attaqué les problèmes de décision de la forme « $Rel(R_1) \subseteq R_2$ », où $R_1$ et $R_2$ sont des langages réguliers d'arbres et *Rel* est une transformation d'arbres. Nous avons étudié ce problème dans le cas où la transformation *Rel* est basée sur un système de réécriture d'arbres. L'intérêt de l'approche était de ré-exprimer la relation *Rel* à l'aide de morphismes et morphismes inverses d'arbres très simples et d'intersection avec un langage régulier. À l'aide de cette nouvelle expression, on garantit la préservation de la régularité des langages et on obtient donc facilement le résultat de décision. Le papier peut aujourd'hui donner des idées de classes de transformations ayant de bonnes caractéristiques pour être candidates pour obtenir des résultat d'apprenabilité.

Franck Seynhaeve est le premier étudiant dont j'ai co-encadré la thèse, avec Sophie Tison. Il a d'abord poursuivi des travaux sur les contraintes ensemblistes, qui étaient le sujet de ma thèse. Avec Franck et Ralf Treinen, nous avons aussi montré une technique qui a permis d'obtenir trois résultats d'indécidabilité. L'un raffinait un joli résultat de Ralf sur la réécriture en un pas, un autre se situait dans le cadre des contraintes ensemblistes et le dernier abordait le problème du vide dans les automates d'arbres à tests d'égalité (Seynhaeve et al., 2001).

### 5.1.3 Résiduels de langages d'arbres

« *Apprendre un langage c'est approcher une congruence de Nerode* » selon J. Engelfriet[2]. Cette citation illustre la proximité entre la théorie des langages et l'apprentissage automatique et plus précisément l'inférence grammaticale. Effectivement, les programmes d'inférence grammaticale reposent sur des propriétés fondamentales des langages. Par exemple, la classe des automates minimaux déterministes est choisie pour représenter les langages réguliers lorsque l'on cherche à les apprendre. Les algorithmes

[2]je crois mais je ne suis plus sûr !

comme RPNI (Oncina and García, 1993) raffinent une partition jusqu'à obtenir les classes d'équivalence de la congruence de Nerode du langage et donc l'automate minimal déterministe.

Un joli résultat de Denis et al. (2002a) s'inscrit dans cette tradition d'exploiter des résultats fondamentaux de la théorie des langages pour inventer de nouveaux algorithmes d'inférence. Le résultat fondamental de Denis et al. (2002a) est d'introduire la classe des automates finis à états résiduels (RFSA). Dans cette classe d'automates non déterministes, il existe un représentant minimal canonique pour chaque langage régulier $L$. Il peut être exponentiellement plus petit que son compagnon déterministe. À chaque état $q$ d'un RFSA reconnaissant $L$, correspond un langage résiduel $L_q$. C'est-à-dire qu'il existe un mot $u$ tel que $L_q = u^{-1}L = \{w \mid uw \in L\}$. Ensuite dans Denis et al. (2002b), les auteurs montrent comment exploiter cette propriété pour bâtir un algorithme d'apprentissage de langages représentés par des automates non déterministes.

La question du déterminisme est importante dans le cadre de l'inférence grammaticale, car la taille de la cible donne une borne minimale de complexité.

Nous avons repris le travail pour établir une correspondance dans les arbres au début de la thèse de Julien. Le travail a été réalisé avec Aurélien Lemay, Alain Terlutte et Rémi Gilleron (Carme et al., 2003). Nous avons défini des automates d'arbres à états résiduels. Toutefois, la correspondance avec le cas des langages de mots n'est pas immédiate. En effet, les automates d'arbres s'entendent selon deux variantes : les automates ascendants et descendants. Si dans la version non déterministe les deux types définissent tous deux la même classe de langages reconnaissables, coïncidant avec les langages réguliers d'arbres, ce n'est pas le cas dans la version déterministe. Les langages reconnus par les automates d'arbres descendant sont strictement inclus dans les réguliers. Les deux variantes se retrouvent aussi dans la définition de résiduels qui correspondent à un état intermédiaire dans un processus de reconnaissance ascendant ou descendant. Dans le premier cas, les résiduels sont des langages de contextes (des arbres à un trou), dans le deuxième ce sont des langages d'arbres.

Les résultats sont semblables à ceux obtenus dans les mots pour le cas des automates d'arbres ascendants : les langages reconnus par les automates d'arbres à états résiduels ascendants sont les langages réguliers. On trouve dans cette classe un représentant minimal canonique pour chaque langage. Dans le cas descendant, il est intéressant de noter l'apparition d'une nouvelle classe de langages strictement comprise entre les réguliers et les langages reconnus par les déterministes descendants.

Du point de vue de l'apprentissage, les algorithmes comme RPNI et l'algorithme se basant sur les résiduels se traduisent immédiatement dans le cas des arbres.

## 5.2 En apprentissage

### 5.2.1 Grammaires catégorielles

J'ai travaillé sur l'apprentissage automatique de grammaires catégorielles lors de la thèse de Daniela Dudau-Sofronie que j'ai co-encadrée avec Isabelle Tellier. Le sujet étant assez différent de ce que j'ai présenté jusqu'ici, je n'en donne que les grandes lignes. Une description plus complète se trouve dans la thèse de Daniela (Sofronie, 2004) et l'habilitation d'Isabelle (Tellier, 2005).

Les grammaires catégorielles sont des grammaires pour les langages de mots. Elles génèrent exactement la classe de langages algébriques. Mais le formalisme catégoriel est tout à fait différent des grammaires algébriques. La donnée d'une grammaire catégorielle est la donnée d'un lexique. Les règles de production sont fixes. À chaque lettre est associée une catégorie dans ce lexique. Les règles définissent un calcul sur les catégories.

Cette propriété d'être lexicalisées donne un statut particulier aux grammaires catégorielles. Elles sont pour cela utilisées pour modéliser les langues naturelles. (De ce fait, on parle plutôt de langages vus comme des ensemble de phrases finies écrites avec des mots, plutôt que d'ensembles de mots finis écrits avec des lettres.) L'apprentissage des grammaires catégorielles a été très bien décrit dans la thèse de Kanazawa (1998). En fait Kanazawa a principalement étudié l'apprentissage de ces grammaires non pas à partir des mots du langage mais des arbres de dérivation de ces mots, privés d'annotations. De ce fait, l'apprentissage se trouve être une formulation un peu cachée de l'apprentissage de langages d'arbres, comme l'ont fait remarquer Besombes and Marion (2004).

Dans le travail de Daniela, sur les idées d'Isabelle, nous avons introduit une information de nature sémantique pour faciliter l'apprentissage de ces grammaires, à partir de mots du langages. Si on se place dans le cadre de l'apprentissage humain de sa langue maternelle, supposer disposer des structures de dérivation des phrases n'est pas très réaliste. On peut par contre défendre l'idée de la présence d'une information sémantique qui va aider le processus d'apprentissage. C'est le principe de compositionnalité qui exprime une relation forte entre syntaxe et sémantique qui permet de justifier l'aide apportée par la sémantique. En termes formels, ce principe se traduit par un isomorphisme entre la structure syntaxique et la structure sémantique des phrases.

Dans le travail de Daniela, nous avons considéré une forme lexicalisée de ce principe de compositionnalité. La catégorie d'un mot peut être écrite sous la forme d'un terme et la sémantique des mots est donnée sous la forme du type de la catégorie. Le type est isomorphe à la catégorie, mais porte une information incomplète. La charge du programme d'apprentissage est de retrouver cette information et d'affecter les catégories aux mots. Les résultats principaux établissent des résultats fondamentaux de ces langages équipés de types dans Dudau-Sofronie et al. (2001) et présente un cadre et un algorithme pour l'apprentissage de ces langages dans Dudau-Sofronie et al. (2003).

### 5.2.2 Arbres de décision alternants

Ce travail réalisé avec Rémi Gilleron et Francesco de Comite est sans doute le premier clairement inscrit dans la thématique de l'apprentissage automatique. Nous avons abordé le problème de la classification supervisée et les techniques de boosting dans un cadre un peu particulier.

Les tâches de classification communément considérées dans la littérature (et aussi dans le chapitre 2 de ce mémoire) sont binaires : le problème est d'associer à des objets décrits dans un espace de représentation exactement une valeur prise dans un ensemble à deux valeurs. On réalise une partition des objets en deux classes. Lorsqu'on réalise une partition des objets en plus de deux classes, comme par exemple associer chaque message à un dossier de messages, on résout un problème multi-classe. Les arbres de décision sont un exemple de procédure de classification binaire ou multi-classe. Un avantage important des arbres de décision est leur lisibilité : il est facile de les interpréter. Enfin, les procédures qui attachent plusieurs étiquettes à chaque description d'objet résolvent des

problèmes multi-étiquettes. Un exemple est l'indexation d'articles de presse qui consiste à leur associer une étiquette dans l'ensemble sport, médecine, économie, divertissement, international. Les articles sur le football ou le cyclisme rentrent par nature dans plusieurs catégories...

Dans les trois cas, on peut envisager l'apprentissage supervisé de ces classifieurs : on connait a priori les classes ou étiquettes, on a accès à des exemples classés ou étiquetés et on doit inférer la procédure en question. L'apprentissage supervisé de classifieurs a été largement étudiée (le lecteur est invité à consulter Cornuéjols and Miclet (2002)). C'est un peu moins le cas des problèmes multi-étiquettes.

Les techniques de boosting sont aussi très populaires dans la communauté de l'apprentissage automatique. Partant d'une question fondamentale des théories formelles de l'apprentissage dans les années 80, résolue au début 90 par le boosting (Freund, 1990; Schapire, 1990), la question[3] a abouti à une solution algorithmique élégante dans le milieu des années 90 et aujourd'hui le boosting est intégré à plusieurs outils commerciaux de fouille de données. La solution algorithmique élégante due à Freund and Schapire (1996) s'appelle Adaboost. Dans de nombreux cas pratiques, le boosting permet d'améliorer les performances d'un programme d'apprentissage. La contre-partie est l'illisibilité des procédures obtenues après boosting.

Nous nous sommes intéressés au problème d'inférence de classifieurs multi-classes et multi-étiquettes, à l'aide des techniques de boosting mais tout en gardant un critère de lisibilité des procédures générées. Le formalisme sous lequel sont représentées les procédures de classification sont des arbres de décision alternants.

Un tel arbre représente un ensemble de règles de décision de la forme :

```
Si condition Alors
   Si Test Alors valeurOui Sinon ValeurNon
```

Les valeurs sont des vecteurs de réels, chaque composante participant à la décision pour une étiquette. L'ensemble des règles, à cause de propriétés des conditions et tests peut se représenter sous la forme d'un arbre. Les conditions et tests sont tels que ces arbres de décision alternants s'appliquent à des objets représentés en attribut-valeur mais aussi à des textes ou aux deux à la fois. Ils peuvent donc s'appliquer à un codage de document `XML` tel que celui utilisé dans le chapitre 2. Le passage d'une représentation d'un objet dans un arbre de décision alternant s'effectue en appliquant toutes les règles et en sommant les vecteurs ainsi calculés. La décision s'applique en regardant le signe de chaque composante.

L'idée des arbres de décision alternants apparaît dans Freund and Mason (1999). Nous l'avons adaptée aux problèmes multi-classe et multi-étiquettes. Nous l'avons aussi appliqué dans le domaine médical à un problème de prédiction du diabète, en collaboration avec des statisticiens et médecins diabétologues (de Comite et al., 2003).

### 5.2.3 Positifs et non étiquetés

Ce travail réalisé en collaboration avec François Denis et Rémi Gilleron montre qu'il est possible d'apprendre des procédures de classification binaires à partir d'exemples

[3]La question fondamentale demande de décrire le cadre formel du problème. Le lecteur trouvera un survol pédagogique dans l'exposé suivant Denis and Gilleron (2000).

d'une seule classe. Un algorithme a été développé et testé. Il illustre expérimentalement la faisabilité de cette approche.

Le qualificatif « supervisée » dans la classification supervisée suppose l'intervention d'un expert qui fournit deux informations : la définition du problème en révélant les classes possibles et un jeu d'exemples déjà classés. Le calcul et l'indication de la classe d'un exemple peut être une opération coûteuse et difficile pour l'expert. Pour minimiser l'intervention de l'expert, sans pour autant la réduire totalement, plusieurs propositions ont été formulées. Certaines ont formé des champs de recherche à part entière comme l'apprentissage actif ou l'apprentissage semi-supervisé.

En apprentissage semi-supervisé, on fait la supposition qu'en plus de données étiquetées, on dispose d'un grand nombre de documents non étiquetés. Cette supposition est totalement justifiée dans bien des cas comme le traitement des spams, la classification de documents sur internet... La présence de documents non étiquetés apporte des informations sur la distribution des exemples, qui peut être exploitée dans des approches statistiques. C'est un champ de recherche très actif comme en témoigne le livre récent de Chapelle et al. (2006) et l'article Zhu (2006). En France, on peut citer les travaux de Massih-Reza Amini et de Yves Grandvalet (Amini and Gallinari, 2005; Grandvalet and Bengio, 2004).

Le travail que j'ai mené avec François et Rémi faisait suite à celui réalisé par Denis et al. (2005). Dans de nombreuses situations, on peut disposer librement de nombreux exemples non étiquetés, et disposer de peu d'exemples d'une classe (les positifs), et pas du tout d'exemples de l'autre classe (les négatifs). C'est le cas par exemple de l'extraction dans le cadre interactif, abordé dans les chapitres précédents. L'utilisateur indique beaucoup plus facilement des exemples de données à extraire mais n'indique jamais des contre-exemples.

La contribution a été de proposer une version de l'algorithme de Bayes naïf pour le cas de la classification à partir de positifs et non étiquetés. L'algorithme de Bayes naïf exploite la définition d'un modèle génératif des données. L'apprentissage consiste essentiellement à retrouver les paramètres de ce modèle. Nous avons montré que nous pouvons réaliser cette approximation des paramètres à l'aide de positifs, de non étiquetés mais aussi de l'estimation de la densité des positifs (Denis et al., 2003). Cette contrainte supplémentaire de disposer de la densité des positifs a été levée dans Magnan (2005); Denis et al. (2006).

Nous avons replacé aussi cet algorithme dans un cadre de co-apprentissage (Blum and Mitchell, 1998) qui consiste à s'aider de deux vues orthogonales sur la donnée pour apprendre un classifieur. L'algorithme (Denis et al., 2003) a été implanté et testé sur des données réelles (WebKB).

# Bibliographie

Amini, M.-R. and Gallinari, P. (2005). Semi-supervised learning with an imperfect supervisor. *Knowledge Information Systems*, 8(4) :385–413.

Besombes, J. and Marion, J.-Y. (2004). Learning reversible categorial grammars from structures. In *Categorial Gramars*.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-

training. In *Proc. 11th Annu. Conf. on Comput. Learning Theory*, pages 92–100. ACM Press, New York, NY.

Carme, J., Gilleron, R., Lemay, A., Terlutte, A., and Tommasi, M. (2003). Residual finite tree automata. In *7th International Conference on Developments in Language Theory*, number 2710 in Lecture Notes in Computer Science, pages 171 – 182. Springer Verlag.

Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. MIT Press.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. Available on : http ://www.grappa.univ-lille3.fr/tata.

Cornuéjols, A. and Miclet, L. (2002). *Apprentissage artificiel ; concepts et algorithmes*. Eyrolles.

Dauchet, M., Tison, S., and Tommasi, M. (2002). Recognizable tree languages and non-linear morphisms. *Theoretical Computer Science*, 281 :219–234. Selected papers in honour of Maurice Nivat.

de Comite, F., Gilleron, R., and Tommasi, M. (2003). Learning multi-label alternating decision trees from texts and data. In *International Conference on Machine Learning and Data Mining*, number 2734 in Lecture Notes in Artificial Intelligence, pages 35–49. Springer Verlag.

Denis, F. and Gilleron, R. (2000). tutoriel sur le boosting. CAP 00.

Denis, F., Gilleron, R., Laurent, A., and Tommasi, M. (2003). Text classification and co-training from positive and unlabeled examples. In *Proceedings of the ICML Workshop : the Continuum from Labeled Data to Unlabeled Data in Machine Learning and Data Mining*, pages 80 – 87.

Denis, F., Gilleron, R., and Letouzey, F. (2005). Learning from positive and unlabeled examples. *Theoretical Computer Science*, 348(1) :70–83.

Denis, F., Lemay, A., and Terlutte, A. (2002a). Residual finite state automata. *Fundamenta Informaticae*, 51(4) :339–368.

Denis, F., Lemay, A., and Terlutte, A. (2002b). Some language classes identifiable in the limit from positive data. In *ICGI 2002*, number 2484 in Lecture Notes in Artificial Intelligence, pages 63–76. Springer Verlag.

Denis, F., Magnan, C., and Ralaivola, L. (2006). Efficient learning of naive bayes classifiers under class-conditional classification noise. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 265–272.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2001). Learning categorial grammars from semantic types. In *proceedings of the 13th Amsterdam Colloquium*, pages 79–84.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2003). A learnable class of classical categorial grammars from typed examples. In *8th Conference on Formal Grammar*, pages 77–88.

Freund, Y. (1990). Boosting a weak learning algorithm by majority. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 202–216, Rochester, New York. ACM Press.

Freund, Y. and Mason, L. (1999). The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann.

Grandvalet, Y. and Bengio, Y. (2004). Semi-supervised learning by entropy minimization. In *Proceedings of NIPS*.

Kanazawa, M. (1998). *Learnable Classes of Categorial Grammars*. The European Association for Logic, Language and Information. CLSI Publications.

Magnan, C. N. (2005). Apprentissage semi-supervisé asymétrique et estimations d'affinités locales dans les protéines. In cois Denis, F., editor, *Actes de CAp 05*, pages 297–312. PUG.

Oncina, J. and García, P. (1993). Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. DSIC-II/47/93.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2) :197–227.

Seynhaeve, F., Tison, S., and Tommasi, M. (1999). Homomorphisms and concurrent term rewriting. In Ciobanu, G. and Paun, G., editors, *Proceedings of the twelfth International Conference on Fundamentals of Computation theory*, number 1684 in Lecture Notes in Computer Science, pages 475–487, Iasi, Romania. Springer Verlag.

Seynhaeve, F., Tison, S., Tommasi, M., and Treinen, R. (2001). Grid structures and undecidable constraint theories. *Theoretical Computer Science*, 258 :453–490.

Sofronie, D. D. (2004). *Apprentissage de grammaires catégorielles pour simuler l'acquisition du langage naturel à l'aide d'informations sémantiques*. PhD thesis, Université Lille 1.

Tellier, I. (2005). Habilitation à diriger des recherches. Université de Lille 3.

Zhu, X. (2006). Semi-supervised learning literature survey. Technical Report Computer Sciences TR 1530, University of Wisconsin. `http://www.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf`.

# Conclusion

Pendant ces quatre années de 2002 à 2006, j'espère avoir mis mes compétences acquises dans les théories formelles des langages d'arbres et l'apprentissage automatique au profit de la création et l'évolution de ce projet Mostrare, de mon entourage et des étudiants dont j'ai pu superviser les travaux.

Mostrare est encore un projet récent, mais qui a donné naissance à de nouvelles ambitions dont Marmota est celle qui m'intéresse le plus en ce moment. Plusieurs thèses qualifiées de Mostrare sont maintenant soutenues ou proches d'être soutenues et j'ai pu participer au co-encadrement de quelques unes d'entres-elles. Les développements logiciels des idées exposées dans ces travaux sont maintenant opérationnels et diffusés. Bien d'autres choses auxquelles je n'ai pas pas participé sont à inscrire au crédit de Mostrare. Elles ne sont pas présentées dans ce mémoire et j'invite le lecteur à consulter le site Internet du projet, les rapports d'activité et le mémoire d'habilitation récent de Jean-Marc Talbot.

En perspectives de mon travail actuel, je sélectionnerais trois points. Le premier est déjà exposé dans la déclaration d'intention du projet Marmota. C'est un travail qui commence, autour des langages probabilistes et des statistiques inférentielles pour les arbres. J'aimerais y contribuer pleinement et j'y trouve des questions fondamentales intéressantes. Deuxièmement, un effort particulier au sein de Mostrare a été mis sur le développement d'applications. Elles peuvent contribuer à mieux nous fédérer et diffuser nos recherches. C'est aussi une démarche que je compte renforcer. Troisièmement, je voudrais réussir à intégrer dans nos algorithmes, une exploitation de connaissances sémantiques de façon à aller supprimer totalement l'intervention de l'utilisateur, tout en gardant une estimation de la qualité des données extraites ou transformées.

# Annexe A

# Bibliographie personnelle et articles

## Revues internationales

Dauchet, M., Tison, S., and Tommasi, M. (2002). Recognizable tree languages and non-linear morphisms. *Theoretical Computer Science*, 281 :219–234. Selected papers in honour of Maurice Nivat.

Gilleron, R., Tison, S., and Tommasi, M. (1999). Set constraints and automata. *Information and Computation*, 149 :1 – 41.

Seynhaeve, F., Tison, S., Tommasi, M., and Treinen, R. (2001). Grid structures and undecidable constraint theories. *Theoretical Computer Science*, 258 :453–490.

## Conférences internationales avec actes et comité de lecture

Carme, J., Gilleron, R., Lemay, A., Terlutte, A., and Tommasi, M. (2003). Residual finite tree automata. In *7th International Conference on Developments in Language Theory*, number 2710 in Lecture Notes in Computer Science, pages 171 – 182. Springer Verlag.

Carme, J., Niehren, J., and Tommasi, M. (2004). Querying unranked trees with stepwise tree automata. In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105 – 118. Springer Verlag.

Caron, A.-C., Seynhaeve, F., Tison, S., and Tommasi, M. (1999). Deciding the satisfiability of quantifier free formulae on one-step rewriting. In Narendran, P. and Rusinowitch, M., editors, *Proceedings of the tenth International Conference on Rewriting Techniques and Applications*, number 1631 in Lecture Notes in Computer Science, pages 103–117, Trento, Italy. Springer Verlag.

Comité, F. D., Gilleron, R., and Tommasi, M. (2001). Learning multi-label alternating decision trees and applications. In Bisson, G., editor, *Proceedings of CAP'01 : Conférence en Apprentissage Automatique*, pages 195–210.

de Comite, F., Gilleron, R., and Tommasi, M. (2003). Learning multi-label alternating decision trees from texts and data. In *International Conference on Machine Learning and Data Mining*, number 2734 in Lecture Notes in Artificial Intelligence, pages 35–49. Springer Verlag.

Denis, F., Gilleron, R., Laurent, A., and Tommasi, M. (2003). Text classification and co-training from positive and unlabeled examples. In *Proceedings of the ICML Workshop : the Continuum from Labeled Data to Unlabeled Data in Machine Learning and Data Mining*, pages 80 – 87.

Denis, F., Gilleron, R., and Tommasi, M. (2002). Text classification from positive and unlabeled examples. In *IPMU'02, 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 1927–1934.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2001a). From logic to grammars via types. In *proceedings of LLL 2001, Learning Language in Logic*, pages 35–46.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2001b). Learning categorial grammars from semantic types. In *proceedings of the 13th Amsterdam Colloquium*, pages 79–84.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2002). A tool for language learning based on categorial grammars and semantic information. In *proceedings of ICGI'2002, International Colloquium on Grammatical Inference (demo session)*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pages 303–305. Springer Verlag.

Dudau-Sofronie, D., Tellier, I., and Tommasi, M. (2003). A learnable class of classical categorial grammars from typed examples. In *8th Conference on Formal Grammar*, pages 77–88.

Gilleron, R., Marty, P., Tommasi, M., and Torre, F. (2006a). Interactive tuples extraction from semi-structured data. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence*.

Gilleron, R., Marty, P., Tommasi, M., and Torre, F. (2006b). Interactive tuples extraction from semi-structured data. In *Proceedings of Web Intelligence*.

Seynhaeve, F., Tison, S., and Tommasi, M. (1999). Homomorphisms and concurrent term rewriting. In Ciobanu, G. and Paun, G., editors, *Proceedings of the twelfth International Conference on Fundamentals of Computation theory*, number 1684 in Lecture Notes in Computer Science, pages 475–487, Iasi, Romania. Springer Verlag.

Seynhaeve, F., Tommasi, M., and Treinen, R. (1997). Grid structures and undecidable constraint theories. In *Proceedings of the $7^{th}$ International Joint Conference on Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 357–368. Springer Verlag.

## Conférences nationales avec actes et comité de lecture

Gilleron, R., Marty, P., Tommasi, M., and Torre, F. (2006). Extraction de relations dans les documents web. In *Revue RNTI - Actes de EGC'06*, pages 415–420.

Jousse, F., Gilleron, R., Tellier, I., and Tommasi, M. (2006). Champs conditionnels aléatoires pour l'annotation d'arbres. In *8ème Conférence francophone sur l'Apprentissage automatique (CAp'2006)*, pages 171–186.

Jousse, F., Tellier, I., Tommasi, M., and Marty, P. (2005). Learning to extract answers in question answering : Experimental studies. In *Actes de CORIA'05*, pages p.85–100. Hermès.

## Divers

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. Available on : http ://www.grappa.univ-lille3.fr/tata.

# Annexe B

# Residual Final State Automata

# Residual Finite Tree Automata

J. Carme, R. Gilleron, A. Lemay, A. Terlutte, and M. Tommasi

Grappa – EA 3588 – Lille 3 University
http://www.grappa.univ-lille3.fr

**Abstract.** Tree automata based algorithms are essential in many fields in computer science such as verification, specification, program analysis. They become also essential for databases with the development of standards such as XML. In this paper, we define new classes of non deterministic tree automata, namely residual finite tree automata (RFTA). In the bottom-up case, we obtain a new characterization of regular tree languages. In the top-down case, we obtain a subclass of regular tree languages which contains the class of languages recognized by deterministic top-down tree automata. RFTA also come with the property of existence of canonical non deterministic tree automata.

## 1 Introduction

The study of tree automata has a long history in computer science; see the survey of Thatcher [Tha73], and the texts of F. Gécseg and M. Steinby [GS84,GS96], and of the TATA group [CDG+97]. With the advent of tree-based metalanguages (SGML and XML) for document grammars, new developments on tree automata formalisms and tree automata based algorithms have been done [MLM01,Nev02]. Also, because of the tree structure of documents, learning algorithms for tree languages have been defined for the tasks of information extraction and information retrieval [Fer02,GK02,LPH00]. We are currently involved in a research project dealing with information extraction systems from semi-structured data. One objective is the definition of classes of tree automata satisfying two properties: there are efficient algorithms for membership and matching, and there are efficient learning algorithms for the corresponding classes of tree languages.

In the present paper, we only consider finite ranked trees. There are bottom-up (also known as frontier to root) tree automata and top-down (also known as root to frontier) tree automata. The top-down version is particularly relevant for some implementations because important properties such as membership[1] can be solved without handling the whole input tree into memory. There are also deterministic tree automata and non-deterministic tree automata. Determinism is important to reach efficiency for membership and other decision properties. It is known that non-deterministic top-down, non-deterministic bottom-up, and deterministic bottom-up tree automata are equally expressive and define regular tree languages. But there is a tradeoff between efficiency and expressiveness because some regular (and even finite) tree languages are not recognized by deterministic top-down tree automata. Moreover, the size of a deterministic bottom-up tree automaton can be exponentially larger than the size of a

[1] given a tree automaton $A$, decide whether an input tree is accepted by $A$.

non-deterministic one recognizing the same tree language. This drawback can be dramatic when the purpose is to build tree automata. This is for instance the case in the problem of tree pattern matching and in machine learning problems like grammatical inference.

The process of learning finite state machines from data is referred as grammatical inference. The first theoretical foundations were given by Gold [Gol67] and first applications were designed in the field of pattern recognition. Grammatical inference mostly focused on learning string languages but recent works are concerned with learning tree languages [Sak90,Fer02,GK02]. In most works, the target tree language is represented by a deterministic bottom-up tree automaton. This is problematic because the time complexity of the learning algorithm depends on the size of the target automaton. Therefore, again it is crucial to define learning algorithms for non-deterministic tree automata. The reader should note that tree patterns [GK02] satisfy this property.

Therefore the aim of this article is to define non-deterministic tree automata corresponding to sufficiently expressive classes of tree languages and having nice properties from the algorithmic viewpoint and from the grammatical inference viewpoint. For this aim, we extend previous works from the string case [DLT02a] to the tree case and we define residual finite state automata (RFTA). The reader should note that learning algorithms for residual finite string automata have been defined [DLT01,DLT02b].

In Section 3, we study the bottom-up case. We define the residual language of a language $L$ w.r.t a ground term $t$ as the set of contexts $c$ such that $c[t]$ is a term in $L$. We define bottom-up residual tree automata as automata whose states correspond to residual languages. Bottom-up residual tree automata are non-deterministic and recognize regular tree languages. We prove that every regular tree language is recognized by a unique canonical bottom-up residual tree automaton, minimal according to the number of states. We give an example of regular tree languages for which the size of the deterministic bottom-up tree automata grows exponentially with respect to the size of the canonical bottom-up residual tree automata.

In Section 4, we study the top-down case. We define the residual language of a language $L$ w.r.t a context $c$ as the set of ground terms $t$ such that $c[t]$ is a term in $L$. We define top-down residual tree automata as automata whose states correspond to residual languages. Top-down residual tree automata are non-deterministic tree automata. Interestingly, the class of languages recognized by top-down residual tree automata is strictly included in the class of regular tree languages and strictly contains the class of languages recognized by deterministic top-down tree automata. We also prove that every tree language in this family is recognized by a unique canonical top-down residual tree automaton; this automaton is minimal according to the number of states.

The definition of residual finite state automata comes with new decision problems. All of them rely on properties of residual languages. It is proved that all residual languages of a given tree language $L$ can be built in both top-down and bottom-up cases. From these constructions we obtain positive answers to decision problems like 'decide whether an automaton is a (canonical) RFTA'. The exact complexity bounds are not given but we conjecture that are identical than in the string case.

The present work is connected with the paper by Nivat and Podelski [NP97]. They consider a monoid framework, whose elements are called pointed trees (contexts in our terminology, special trees in [Tho84]), to define tree automata. They define a Nerode congruence in the bottom-up case and in the top-down case. Their work leads to the generalization of the notion of deterministic to l-r-deterministic (context-deterministic in our terminology) for top-down tree automata. They have a minimization procedure for this class of automata. It should be noted that the class of languages recognized by context-deterministic tree automata (also called homogeneous tree languages) is strictly included in the class of languages recognized by residual top-down tree automata.

## 2 Preliminaries

We assume that the reader is familiar with basic knowledge about tree automata. We follow the notations defined in TATA [CDG$^+$97].

A ranked alphabet is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into $\mathbb{N}$. The set of symbols of arity $p$ is denoted by $\mathcal{F}_p$. The set of *terms* over $\mathcal{F}$ is denoted by $\mathcal{T}(\mathcal{F})$. Let $\diamond$ be a special constant which is not in $\mathcal{F}$. The set of *contexts* (also known as pointed trees in [NP97] and special trees in [Tho84]), denoted by $\mathcal{C}(\mathcal{F})$, is the set of terms which contains exactly one occurrence of $\diamond$. The expression $c[\diamond]$ denotes a context, we only write $c$ when there is no ambiguity. We denote by $c[t]$ the term obtained from $c[\diamond]$ by replacing $\diamond$ by a term $t$.

A *bottom-up Finite Tree Automaton* ($\uparrow$-FTA) over $\mathcal{F}$ is a tuple $A = (Q, \mathcal{F}, Q_f, \Delta)$ where $Q$ is a finite set of states, $Q_f \subseteq Q$ is a set of final states, and $\Delta$ is a set of transition rules of the form $f(q_1, \ldots, q_n) \to q$ where $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$. In this paper, the size of an automaton refers to its size in number of states, so two automaton which have the same number of states but different number of rules are considered as having the same size. When $n = 0$ a rule is written $a \to q$, where $a$ is a constant. The *move relation* is written $\to_A$ and $\to_A^*$ is the reflexive and transitive closure of $\to_A$. A term $t$ reaches a state $q$ if and only if $t \to_A^* q$. A state $q$ *accepts* a context $c$ if and only if there exists a $q_f \in Q_f$ such that $c[q] \to_A^* q_f$. The automaton $A$ recognizes a term $t$ if and only if there exists a $q_f \in Q_f$ such that $t \to_A^* q_f$. The language recognized by $A$ is the set of all terms recognized by $A$, and is denoted by $L(A)$.

Two $\uparrow$-FTA are equivalent if they recognize the same tree language. A $\uparrow$-FTA $A = (Q, \mathcal{F}, Q_f, \Delta)$ is *trimmed* if and only if all its states can be reached by at least one term and accepts at least one context. A $\uparrow$-FTA is *deterministic* ($\uparrow$-DFTA) if and only if there are no two rules with the same left-hand side in its set of rules. A tree language is *regular* if and only if it is recognized by a bottom-up tree automaton. As any $\uparrow$-FTA can be changed into an equivalent trimmed $\uparrow$-DFTA, any regular tree language can be recognized by a trimmed $\uparrow$-DFTA.

Let $L$ be a tree language over a ranked alphabet $\mathcal{F}$ and $t$ a term. The bottom-up residual language of $L$ relative to a term $t$, denoted by $t^{-1}L$, is the set of all contexts in $\mathcal{C}(\mathcal{F})$ such that $c[t] \in L$: $t^{-1}L = \{c \in \mathcal{C}(\mathcal{F}) \mid c[t] \in L\}$.

Note that a bottom-up residual language is a set of contexts, and not a tree language. The Myhill-Nerode congruence for tree languages can be defined by

two terms $t$ and $t'$ are equivalent if they define the same residual languages. From the Myhill-Nerode theorem for tree languages, we get the following result: a tree language is recognizable if and only if the number of residual languages is finite.

A *top-down finite tree automaton* ($\downarrow$-FTA) over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ where $Q$ is a set of states, $I \subseteq Q$ is a set of initial states, and $\Delta$ is a set of rewrite rules of the form $q(f) \to f(q_1, \ldots, q_n)$ where $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$. Again, if $n = 0$ the rule is written $q(a) \to a$. The *move relation* is written $\to_A$ and $\to_A^*$ is the reflexive and transitive closure of $\to_A$. A state $q$ accepts a term $t$ if and only if $q(t) \to_A^* t$. $A$ recognizes a term $t$ if and only if at least one of its initial states accepts it. The language recognized by $A$ is the set of all ground terms recognized by $A$ and is denoted by $L(A)$. Any regular tree language can be recognized by a $\downarrow$-FTA. This means that $\downarrow$-FTA and $\uparrow$-FTA have the same expressive power. A $\downarrow$-FTA is *deterministic* ($\downarrow$-DFTA) if and only if its set of rules does not contain two rules with the same left-hand side. Unlike $\uparrow$-DFTA, $\downarrow$-DFTA are not able to recognize all regular tree languages.

Let $L$ be a tree language over a ranked alphabet $\mathcal{F}$, and $c$ a context of $\mathcal{C}(\mathcal{F})$. The top-down residual language of $L$ relative to $c$, denoted by $c^{-1}L$, is the set of ground terms $t$ such that $c[t] \in L$: $c^{-1}L = \{t \in \mathcal{T}(\mathcal{F}) \mid c[t] \in L\}$.

The definition of top-down residual languages comes with an equivalence relation on contexts. It is worth noting that it does not define a congruence over terms. Nonetheless, based on [NP97], it can be shown that a tree language $L$ is regular if and only if the number of top-down residual languages associated with $L$ is finite. In the proof, it is used that the number top-down residual languages is lower than the number of bottom-up residual languages.

The full proofs of theorems and propositions are given in [CGL$^+$03].

## 3 Bottom-up residual finite tree automata

In this section, we introduce a new class of bottom-up finite tree automata, called bottom-up residual finite tree automata ($\uparrow$-RFTA). This class of automata shares some interesting properties with both bottom-up deterministic and non-deterministic finite tree automata which both recognize the class of regular tree languages.

On the one hand, as $\uparrow$-DFTA, $\uparrow$-RFTA admits a unique canonical form, based on a correspondence between states and residual languages, whereas $\uparrow$-FTA does not. On the other hand, $\uparrow$-RFTA are non-deterministic and can be much smaller in their canonical form than their deterministic counter-parts.

### 3.1 Definition and expressive power of bottom-up residual finite tree automata

First, let us precise the nature of this correspondence, then let us give the formal definition of $\uparrow$-residual tree automata and describe their properties.

In order to establish the nature of this correspondence between states and residual languages, let us introduce the notion of state languages. The *state language* $C_q$ of a state $q$ is the set of contexts accepted by the state $q$:

$$C_q = \{c \in \mathcal{C}(\mathcal{F}) \mid \exists q_f \in Q_f, c[q] \rightarrow_A^* q_f\}.$$

As shown by the following example, state languages are generally not residual languages:

*Example 1.* Consider the tree language $L = \{f(a_1, b_1), f(a_1, b_2), f(a_2, b_2)\}$ over $\mathcal{F} = \{f(,), a_1, b_1, a_2, b_2\}$. This language $L$ is recognized by the tree automaton $A = (\{q_1, q_2, q_3, q_4, q_5\}, \mathcal{F}, \{q_5\}, \Delta)$ where $\Delta = \{a_1 \rightarrow q_1, b_1 \rightarrow q_2, b_2 \rightarrow q_3, a_2 \rightarrow q_4, a_1 \rightarrow q_4, f(q_1, q_2) \rightarrow q_5, f(q_4, q_3) \rightarrow q_5\}$. Residual languages of $L$ are $a_1^{-1}L = \{f(\diamond, b_1), f(\diamond, b_2)\}$, $b_1^{-1}L = \{f(a_1, \diamond)\}$, $b_2^{-1}L = \{f(a_1, \diamond), f(a_2, \diamond)\}$, $a_2^{-1}L = \{f(\diamond, b_2)\}$, $f(a_1, b_1)^{-1}L = \{\diamond\}$. The state language of $q_1$ is $\{f(\diamond, b_1)\}$, which is not a residual language. The tree $a_1$ reaches $q_1$, so each context accepted by $q_1$ is an element of the residual language $a_1^{-1}L$, which means that $C_{q_1} \subset a_1^{-1}L$. But the reverse inclusion is not true because$f(\diamond, b_2)$ is not an element of $C_{q_1}$. The reader should note that this situation is possible because $A$ is non-deterministic.

In fact, it can be proved (the proof is omitted) that residual languages are unions of state languages. For any $L$ recognized by a tree automaton $A$, we have

$$\forall t \in T(\mathcal{F}), t^{-1}L = \bigcup_{q \in Q,\ t \rightarrow_A^* q} C_q. \tag{1}$$

As a consequence, if $A$ is deterministic and trimmed, each residual language is a state language and conversely.

We can define a new class of non-deterministic automata stating that each state language must correspond to a residual tree language. We have seen that residual tree languages are related to the Myhill-Nerode congruence and we will show that minimization of tree automata can be extended in the definition of a canonical form for this class of non-deterministic tree automata.

**Definition 1.** *A bottom-up residual tree automaton (↑-RFTA) is a ↑-FTA* $A = (Q, \mathcal{F}, Q_f, \Delta)$ *such that* $\forall q \in Q$, $\exists t \in T(\mathcal{F})$, $C_q = t^{-1}L(A)$.

According to the above definition and previous remarks, it can be shown that every trimmed ↑-DFTA is a ↑-RFTA. As a consequence, ↑-RFTA have the same expressive power than finite tree automata:

**Theorem 1.** *The class of tree languages recognized by ↑-RFTA is the class of regular tree languages.*

As an advantage of ↑-RFTA, the number of states of an ↑-RFTA can be much smaller than the number of states of any equivalent ↑-DFTA:

**Proposition 1.** *There exists a sequence* $(L_n)$ *of regular tree languages such that for each* $L_n$*, the size of the smallest ↑-DFTA which recognizes* $L_n$ *is an exponential function of* $n$*, and the size of the smallest ↑-RFTA which recognizes* $L_n$ *is a linear function of* $n$.

*Sketch of proof* We give an example of regular tree languages for which the size of the $\uparrow$-DFTA grows exponentially with respect to the size of the equivalent canonical $\uparrow$-RFTA. A path is a sequence of symbols from the root to a leaf of a tree. The length of a path is the number of symbols on the path, except the root. Let $\mathcal{F} = \{f(,), a\}$ and let us consider the tree language $L_n$ which contains exactly the trees with at least one path of length $n$. Let $A_n = (Q, \mathcal{F}, Q_f, \Delta)$ be a $\uparrow$-FTA defined by: $Q = \{q_*, q_0, \ldots, q_n\}, Q_f = \{q_0\}$ and

$$\Delta = \{a \to q_*, a \to q_n, f(q_*, q_*) \to q_*\} \cup \bigcup_{k \in [1,\ldots,n], q \in Q \setminus \{q_0\}}^{n} \{f(q_k, q) \to q_{k-1}, f(q, q_k) \to q_{k-1}, f(q_k, q) \to q_*, f(q, q_k) \to q_*\}$$

Let $C_*$ be the set of contexts which contain at least one path of length $n$. Let $C_i$ be the set of contexts whose path from the root to $\diamond$ is of length $i$. Let $t_*$ be a term such that all its paths are of length greater than $n$. Note that the set of contexts $c$ such that $c[t_*]$ belongs to $L_n$ is exactly the set of contexts $C_*$. Let $t_0 \ldots t_n$ be terms such that for all $i \leq n$, $t_i$ contains exactly one path of length smaller than $n$, and the length of this path is $n - i$. Therefore, $t_i^{-1} L_n$ is the set of contexts $C_* \cup C_i$.

One can verify that $C_{q_*}$ is exactly $t_*^{-1} L_n = C_*$, and for all $i \leq n$, $C_{q_i}$ is exactly $t_i^{-1} L_n = C_* \cup C_i$. The reader should note that rules of the form $f(q_k, q) \to q_*$ and $f(q, q_k) \to q_*$ are not useful to recognize $L_n$ but they are required to obtain a $\uparrow$-RFTA (because $C_i$ is not a residual language of $L_n$). So $A_n$ is a $\uparrow$-RFTA and recognizes $L_n$. The size of $A_n$ is $n + 2$.

The construction of the smallest $\uparrow$-DFTA which recognizes $L(A_n)$ is left to the reader. But, it can easily be shown that the number of states is in $O(2^n)$ because states must store lengths of all paths smaller than $n$. □

Unfortunately, the size of a $\uparrow$-RFTA can be exponentially larger than the size of an equivalent $\uparrow$-FTA.

### 3.2 The canonical form of bottom-up residual tree automata

As $\uparrow$-DFTA, $\uparrow$-RFTA have the interesting property to admit a canonical form. In the case of $\uparrow$-DFTA, there is a one-to-one correspondence between residual languages and state languages. This is a consequence of the Myhill-Nerode theorem for trees.

A similar result holds for $\uparrow$-RFTA. In a canonical $\uparrow$-RFTA, the set of states is in one-to-one correspondence with a subset of residual languages called prime residual languages.

**Definition 2.** *Let $L$ be a tree language. A bottom-up residual language of $L$ is* composite *if and only if it is the union of the bottom-up residual languages that it strictly contains:*

$$t^{-1} L = \bigcup_{t'^{-1} L \subsetneq t^{-1} L} t'^{-1} L.$$

*A residual language is* prime *if and only if it is not composite.*

*Example 2.* Let us consider again the tree languages in the proof of Proposition 1. Let $Q_n$ be the set of states of $A_n$. All the $n+2$ states $q_*, q_0, \ldots, q_n$ of $Q_n$ have state languages which are prime residual languages. The subset construction applied on $A_n$ to build a $\uparrow$-DFTA $D_n$ leads to consider states which are subsets of $Q$. The state language of a state $\{q_{k_1} \ldots q_{k_n}\}$ is a composite residual language. It is the union of $t_{q_{k_1}}^{-1} L \ldots t_{q_{k_n}}^{-1} L$.

In canonical $\uparrow$-RFTAs, all state languages are prime residual languages.

**Theorem 2.** *Let $L$ be a regular tree language and let us consider the $\uparrow$-FTA $A_{can} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by:*

- *$Q$ is in bijection with the set of all prime bottom-up residual languages of $L$. We denote by $t_q$ a ground term such that $q$ is associated with $t_q^{-1}L$ in this bijection*
- *$Q_f$ is the set of all elements $q$ of $Q$ such that $t_q^{-1}L$ contains the void context $\diamond$,*
- *$\Delta$ contains all the rules $f(q_1, \ldots, q_n) \to q$ such that $t_q^{-1}L \subseteq (f(t_{q_1}, \ldots, t_{q_n}))^{-1}L$ and all the rules $a \to q$ such that $a \in \mathcal{F}_0$ and $t_q^{-1}L \subseteq a^{-1}L$.*

*$A_{can}$ is a $\uparrow$-RFTA, it is the smallest $\uparrow$-RFTA in number of states which recognizes $L$, and it is unique up to a renaming of its states.*

*Sketch of proof* There are three things to prove in this theorem: the canonical $\uparrow$-RFTA $A_{can} = (Q, \mathcal{F}, Q_f, \Delta)$ of a regular tree language $L$ recognizes $L$, it is a $\uparrow$-RFTA, and there cannot be any strictly smaller $\uparrow$-RFTA which recognizes $L$. The three points are proved in this order.

We first have to prove the equality $L(A_{can}) = L$. It follows from the identity $(\circledast)$ $\forall t,\ t^{-1}L = \bigcup_{q \in Q,\, t \to_{A_{can}}^* q} t_q^{-1}L$ which can be proved inductively on the height of $t$. Using this property, we have:

$$t \in L \Leftrightarrow \diamond \in t^{-1}L \underset{\circledast}{\Leftrightarrow} \diamond \in \bigcup_{q \in Q,\, t \to_{A_{can}}^* q} t_q^{-1}L \Leftrightarrow \exists q_f \in Q_f, t \to_{A_{can}}^* q_f \Leftrightarrow t \in L(A_{can})$$

The equality between $L$ and $L(A_{can})$ helps us to prove the characterization of $\uparrow$-RFTA: $t_q^{-1}L = C_q^{A_{can}}$ where $C_q^{A_{can}}$ is the state language of $q$ in $A_{can}$.

The last point can be proved in such a way. In a $\uparrow$-RFTA, any residual language is a union of state languages, and any state language is a residual language. So any prime residual language is a state language, so there is at least as much states in a $\uparrow$-RFTA as prime residual languages admitted by its corresponding tree language.

□

The canonical automaton is uniquely defined by the tree language under consideration, but there may be other automata which have the same number of states. The canonical $\uparrow$-RFTA is unique because it has the maximum number of rules. Even though all its states are associated to prime residual languages, the automaton considered in the proof of Proposition 1 is not the canonical one because some rules are missing: $\bigcup_{k=1}^{n} \{f(q_k, q_0) \to q_{k-1}, f(q_0, q_k) \to q_{k-1}\}$ and $\bigcup_{q \in Q} \{f(q, q_0) \to q_*, f(q, q_0) \to q_*\}$.

# 4 Top-Down residual finite tree automata

The definition of top-down residual finite tree automata (↓-RFTA) is tightly correlated with the definition of ↑-RFTA. Similarly to ↑-RFTA, ↓-RFTA are defined as non-deterministic tree automata where each state language is a residual language. Any ↓-RFTA can be transformed in a canonical equivalent ↓-RFTA — minimal in the number of states and unique up to state renaming.

The main difference between the bottom-up and the top-down case is in the problem of the expressive power of tree automata. The three classes of bottom-up tree automata, ↑-DFTA, ↑-RFTA or ↑-FTA, have the same expressive power. In the top-down case, deterministic, residual and non-deterministic tree automata have different expressive power. This makes the canonical form of ↓-RFTA more interesting. Compared to the minimal form of ↓-DFTA, it can be smaller when both exist, and it exists for a wider class of tree languages.

Let us introduce ↓-RFTA through their similarity with ↑-RFTA, then study this specific problem of expressiveness.

## 4.1 Analogy with bottom-up residual tree automata

Let us formally define state languages in the top-down case:

**Definition 3.** *Let $L$ be a regular tree language over a ranked alphabet $\mathcal{F}$, let $A$ be a top-down tree automaton which recognizes $L$, and let $q$ be a state of this automaton. The state language of $L$ relative to $q$, written $L_q$, is the set of terms which are accepted by $q$:*

$$L_q = \{t \in \mathcal{T}(\mathcal{F}) \mid q(t) \rightarrow_A^* t\}.$$

It follows from this definition some properties similar to those already studied in the previous section. Firstly, state languages are generally not residual languages. Secondly, residual languages are unions of state languages. Let us define $Q_c$:

$$Q_c = \{q \mid q \in Q, \exists q_i \in I, q_i(c[\diamond]) \rightarrow_A^* c[q(\diamond)]\}.$$

We have the following relation between state languages and residual languages.

**Lemma 1.** *Let $L$ be a tree language and let $A = (Q, \mathcal{F}, I, \Delta)$ be a top-down tree automaton which recognizes $L$. Then $\forall c \in \mathcal{C}(\mathcal{F}), \bigcup_{q \in Q_c} L_q = c^{-1}L$.*

These similarities lead us to this definition of top-down residual tree automata:

**Definition 4.** *A top-down Residual Finite Tree Automaton (↓-RFTA) recognizing a tree language $L$ is a ↓-FTA $A = (Q, \mathcal{F}, I, \Delta)$ such that: $\forall q \in Q$, $\exists c \in \mathcal{C}(\mathcal{F})$, $L_q = c^{-1}L$.*

Languages defined in the proof of Proposition 1 are still interesting here to define examples of top-down residual tree automata:

*Example 3.* Let us consider again the family of tree languages $L_n$, and the family of corresponding $\uparrow$-RFTA $A_n$. For every $n$, let $A'_n$ be the $\downarrow$-RFTA defined by: $Q = \{q_*, q_0, \ldots, q_n\}, Q_i = \{q_0\}$ and $\Delta = \{q_*(a) \to a, q_n(a) \to a, q_*(f) \to f(q_*, q_*)\} \cup \bigcup_{k=1}^{n} \{q_{k-1}(f) \to f(q_k, q_*), q_{k-1}(f) \to f(q_*, q_k)\}$.

For every $k \leq n$, the state language of $q_k$ is equal to $L_{n-k}$. And, $L_{n-k}$ is the top-down residual language of $c_k$, where $c_k$ is a context whose height from the root to the special constant $\diamond$ is $k$ and $c_k$ does not contain any path whose length is smaller or equal to $n$. The state language of $q_*$ is $\mathcal{T}(\mathcal{F})$. And, $\mathcal{T}(\mathcal{F})$ is the top-down residual language of $L_n$ relative to $c_*$, where $c_*$ is a context who contains a path whose length is $n$. So $A'_n$ is a $\downarrow$-RFTA. Moreover, it is easy to verify that $A'_n$ recognizes $L_n$.

### 4.2 The expressive power of top-down tree automata

*Top-down deterministic automata and path-closed languages* A tree language $L$ is *path-closed* if:

$$\forall c \in C(\mathcal{F}), c[f(t_1, t_2)] \in L \wedge c[f(t'_1, t'_2)] \in L \Rightarrow c[f(t_1, t'_2)] \in L.$$

The reader should note that the definition only considers binary symbols, the definition can easily be extended to $n$-ary symbols. The class of languages that $\downarrow$-DFTA can recognize is the class of path-closed languages [Vir81].

*Context-deterministic automata and homogeneous languages.* Podelski and Nivat in [NP97] have defined *l-r-deterministic* top-down tree automata. In the present paper, let us call them top-down *context-deterministic* tree automata.

**Definition 5.** *A top-down context-deterministic tree automaton ($\downarrow$-CFTA) $A$ is a $\downarrow$-FTA such that for every context $c \in \mathcal{C}(\mathcal{F})$, $Q_c$ is either the empty set or a singleton set.*

An *homogeneous language* is a tree language $L$ satisfying:

$$\forall c \in C(\mathcal{F}), c[f(t_1, t_2)] \in L \wedge c[f(t_1, t'_2)] \in L \wedge c[f(t'_1, t_2)] \Rightarrow c[f(t'_1, t'_2)] \in L.$$

Again, the definition can easily be extended from the binary case to $n$-ary symbols. They have shown that the class of languages recognized by $\downarrow$-CFTA is the class of homogeneous languages.

*The hierarchy* A $\downarrow$-DFTA is a $\downarrow$-CFTA. For $\downarrow$-CFTA and $\downarrow$-RFTA, we have the following result:

**Lemma 2.** *Any trimmed $\downarrow$-CFTA is a $\downarrow$-RFTA.*

*Proof.* Let $A = (Q, \mathcal{F}, I, \Delta)$ be a trimmed $\downarrow$-CFTA recognizing a tree language $L$. As $A$ is trimmed, all states are reachable, so for every $q$, there exists a $c$ such that $q \in Q_c$. Then, by definition of a $\downarrow$-CFTA, for every $q$, there exists a $c$ such that $\{q\} = Q_c$. Using Lemma 1, we have:

$$\forall q \in Q, \exists c \in \mathcal{C}(\mathcal{F}), L_q = c^{-1}L.$$

stating that $A$ is a $\downarrow$-RFTA. □

Therefore, if we denote by $\mathcal{L}_{\mathcal{C}}$ the class of tree languages recognized by a class of automata $\mathcal{C}$, we obtain the following hierarchy:

$$\mathcal{L}_{\downarrow-DFTA} \subseteq \mathcal{L}_{\downarrow-CFTA} \subseteq \mathcal{L}_{\downarrow-RFTA} \subseteq \mathcal{L}_{\downarrow-FTA}$$

*The hierarchy is strict*

- Let $L = \{f(a,b), f(b,a)\}$. $L_1$ is homogeneous but not path-closed. Therefore $L$ can be recognized by a $\downarrow$-CFTA, but can not be recognized by a $\downarrow$-DFTA.
- The tree languages $L_n$ in the proof of Proposition 1 are not recognized by $\downarrow$-CFTA. We can easily verify that $L_n$ is not homogeneous. Indeed, if $t$ is a term which has a path whose length is equal to $n-1$, and $t'$ a term which does not have any path whose length is smaller than $n$, $f(t,t)$, $f(t,t')$, $f(t',t)$ belong to $L_n$, but $f(t',t')$ does not. And, we have already shown that $L_n$ is recognized by a $\downarrow$-RFTA.
- Let $L' = \{f(a,b), f(a,c), f(b,a), f(b,c), f(c,a), f(c,b)\}$. $L'$ is a finite language, therefore it is a regular tree language which can be recognized by a $\downarrow$-FTA. $L'$ cannot be recognized by a $\downarrow$-RFTA. To prove that, let us consider $A'$ a $\downarrow$-FTA which recognizes $L'$. The top-down residual languages of $L'$ are $\{a,b\}$, $\{a,c\}$, $\{b,c\}$ and $L'$. As $A'$ recognizes $L'$, it recognizes $f(a,b)$. This implies the existence of three states $q_1$, $q_2$, $q_3$ and three rules $q_1(f) \to f(q_2,q_3)$, $q_2(a) \to a$, and $q_3(b) \to b$. If $A'$ was a $\downarrow$-RFTA, then $q_2$ would accept a residual language. As $q_2$ accepts $a$, it would accept either $\{a,b\}$ or $\{a,c\}$. Similarly, $q_3$ would accept either $\{a,b\}$ or $\{b,c\}$. In these conditions, and thanks to the rule $q_1(f) \to f(q_2,q_3)$, $A'$ would recognize $f(a,a)$, $f(b,b)$ or $f(c,c)$. So $A'$ cannot be a $\downarrow$-RFTA.

Therefore, we obtain the following result:

**Theorem 3.** $\mathcal{L}_{\downarrow-DFTA} \subsetneq \mathcal{L}_{\downarrow-CFTA} \subsetneq \mathcal{L}_{\downarrow-RFTA} \subsetneq \mathcal{L}_{\downarrow-FTA}$

So top-down residual tree automata are strictly more expressive than context-deterministic tree automata. But as far as we know, there is no straightforward characterization of the tree languages recognized by $\downarrow$-RFTA.

### 4.3 The canonical form of top-down residual tree automata

The problem of the canonical form of top-down tree automata is similar to the bottom-up case. Whereas there is no way to reduce a non-deterministic top-down tree automaton to a unique canonical form, a top-down residual tree automaton can take such a form. Its definition is similar to the definition of the canonical bottom-up tree automaton.

In the same way that we have defined composite bottom-up residual language, a top-down residual language of $L$ is *composite* if and only if it is the union of the top-down residual languages that it strictly contains and a residual language is *prime* if and only if it is not composite.

**Theorem 4.** *Let $L$ be a tree language in the class $\mathcal{L}_{\downarrow-RFTA}$. Let us consider the $\downarrow$-RFTA $A_{can} = (Q, \mathcal{F}, I, \Delta)$ defined by:*

- *$Q$ is a set of state in bijection with the prime residual languages of $L$. For each of these residual languages, there exists a $c_q$ such that $q$ is associated with $c_q^{-1}L$ in this bijection.*
- *$I$ is the set of prime residuals which are subsets of $L$.*
- *$\Delta$ contains all the rules $q(a) \to a$ such that $a$ is a constant and $c_q[a] \in L$, and all the rules $q(f) \to f(q_1, \ldots, q_n)$ such that for all $t_1 \ldots t_n$ where $t_i \in c_{q_i}^{-1}L$, $c_q[f(t_1, \ldots, t_n)] \in L$.*

*$A_{can}$ is a $\downarrow$-RFTA, it is the smallest $\downarrow$-RFTA in number of states which recognizes $L$, and it is unique up to a renaming of its states.*

*Sketch of proof*

The proof is mainly based on this lemma: $t \in c_q^{-1}L \Leftrightarrow t \in L_q^{A_{can}}$
where $L_q^{A_{can}}$ is the state language of $q$ in the automaton $A_{can}$.

This lemma is proved by induction on the height of $t$. This is not a straightforward induction. It involves the rules of a $\downarrow$-RFTA automaton $A'$ which recognizes $L$. Its existence is granted by the hypothesis of the theorem. Once this is proved, it can be easily deduced that $A_{can}$ recognizes $L$ and is a RFTA. As there is one state per prime residual in $A_{can}$, it is minimal in number of states.

□

## 5 Decidability issues

Some decision problems naturally arise with the definition of RFTA. Most of these problems are solved just noting that one can build all residual languages of a given regular language $L$ defined by a non-deterministic tree automaton. In the bottom-up case, the state languages of the minimal $\uparrow$-DFTA which recognizes $L$ are exactly the bottom-up residual languages of $L$, and this automaton can be built with the subset construction. In the top-down case, the subset construction does not necessarily gives us an automaton which recognizes exactly $L$, but there exists a way to construct the top-down residual languages of a tree language $L$ described in [CGL$^+$03]. Therefore, knowing whether a tree automaton is a RFTA, whether a residual language is prime or composite, and whether a tree automaton is a canonical RFTA are decidable. These problems have not been deeply studied in terms of complexity, but they are at least as hard as the similar problems with strings, that is they are PSPACE-hard ([DLT02a]).

## 6 Conclusion

We have defined new classes of non-deterministic tree automata. In the bottom-up case, we get another characterization of regular tree languages. More interestingly, in the top-down case, we obtain a subclass of the regular tree languages. For both cases, we have a canonical form and the size of residual tree automata can be much smaller than equivalent (when exist) deterministic ones.

We are currently extending these results to the case of unranked trees because our application domain is concerned with html and xml documents. Also, we are designing learning algorithms for residual finite tree automata extending previous algorithms for residual finite string automata [DLT01,DLT02b].

**Acknowledgements** The authors wish to thank the anonymous reviewers for their critisms and suggestions. This research was partially supported by "TACT-TIC" région Nord-Pas-de-Calais — FEDER and the MOSTRARE INRIA project.

## References

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.

[CGL+03] J. Carme, R. Gilleron, A. Lemay, A. Terlutte, and M. Tommasi. Residual finite tree automata. Technical report, GRAPPA, 2003.

[DLT01] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using rfsa. In *ALT 2001*, number 2225 in Lecture Notes in Artificial Intelligence. Springer Verlag, 2001.

[DLT02a] F. Denis, A. Lemay, and A. Terlutte. Residual finite state automata. *Fundamenta Informaticae*, 51(4):339–368, 2002.

[DLT02b] F. Denis, A. Lemay, and A. Terlutte. some language classes identifiable in the limit from positive data. In *ICGI 2002*, number 2484 in Lecture Notes in Artificial Intelligence, pages 63–76. Springer Verlag, 2002.

[Fer02] Henning Fernau. Learning tree languages from text. In *Proc. 15th Annual Conference on Computational Learning Theory, COLT 2002*, pages 153 – 168, 2002.

[GK02] Sally A. Goldman and Stephen S. Kwek. On learning unions of pattern languages and tree patterns in the mistake bound model. *Theorical Computer Science*, 288(2):237 – 254, 2002.

[Gol67] E.M. Gold. Language identification in the limit. *Inform. Control*, 10:447–474, 1967.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akademiai Kiado, 1984.

[GS96] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag, 1996.

[LPH00] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *ICDE*, pages 611–621, 2000.

[MLM01] M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, 2001.

[Nev02] F. Neven. Automata, xml and logic. In *Proceedings of CSL*, pages 2–26, 2002.

[NP97] M. Nivat and A. Podelski. Minimal ascending and descending tree automata. *SIAM Journal on Computing*, 26(1):39–58, February 1997.

[Sak90] Yasubumi Sakakibara. learning context-free grammars from structural data in polynomial time. *Theorical Computer Science*, 76:223 – 242, 1990.

[Tha73] J.W. Thatcher. Tree automata: an informal survey. In A.V. Aho, editor, *Currents in the theory of computing*, pages 143–178. Prentice Hall, 1973.

[Tho84] Wolfgang Thomas. Logical aspects in the study of tree languages. In *Proceedings of the 9th International Colloquium on Trees in Algebra and Programming, CAAP '84*, pages 31 – 50, 1984.

[Vir81] J. Viragh. Deterministic ascending tree automata. *Acta Cybernetica*, 5:33–42, 1981.

# Annexe C

# Querying Unranked Trees with Stepwise Tree Automata

# Querying Unranked Trees with Stepwise Tree Automata

Julien Carme Joachim Niehren Marc Tommasi

Mostrare project, INRIA Futurs, Lille, France

**Abstract.** The problem of selecting nodes in unranked trees is the most basic querying problem for XML. We propose *stepwise tree automata* for querying unranked trees. Stepwise tree automata can express the same monadic queries as monadic Datalog and monadic second-order logic. We prove this result by reduction to the ranked case, via a new systematic correspondence that relates unranked and ranked queries.

## 1 Introduction

Querying semi-structured documents is a base operation for information extraction from the Web or semi-structured databases. It requires expressive query languages whose queries can be answered efficiently [8]. The most widely known querying language these days is the W3C standard XPath (see e.g. [10, 9]).

Semi-structured documents in XML or HTML form *unranked trees* whose nodes may have an unbounded list of children. The most basic querying problem is to select sets of nodes in unranked trees. *Monadic queries* approach this problem declaratively. They specify sets of nodes in a tree that can then be computed by a generic algorithm.

We are interested in query languages that can describe all regular sets of nodes in trees. This property is satisfied by three classes of queries, those represented by tree automata [16, 12, 3, 13, 6], *monadic second-order logic* (MSO) [16, 8] and *monadic Datalog* [1, 7] over trees. Automata and Datalog queries can be answered in linear time. They are satisfactory in efficiency and expressiveness, in theory and practice [11].

Unranked trees are problematic in that they may be recursive in depth and breadth, in contrast to ranked trees. This additional level of recursion needs to be accounted for by recursive queries. In MSO and monadic Datalog, breadth recursion can be programmed from the next_sibling relation. Unfortunately, this relation cannot be expressed in WS$\omega$S, so that traditional results on ranked trees don't carry over for free. Selection automata [6] reduce breadth recursion to depth recursion, by operating on binary encodings of unranked trees. Encodings are problematic in that they alter locality and path properties; furthermore the close relationship to unranked Datalog queries gets lost. Hedge automata [16, 12, 3] express horizontal recursion by an extra recursion level in transition rules. This syntactic extension leads to numerous technical problems [13, 7] that one might prefer to avoid.

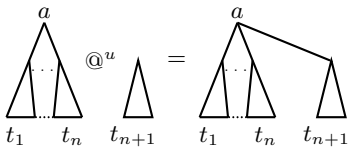



**Fig. 1.** Tree extension

In this paper, we propose *stepwise tree automata* for querying unranked trees. Stepwise tree automata are traditional tree automata that can either operate on unranked or ranked trees. They combine the advantages of selection and hedge automata. They model horizontal recursion by traversing siblings stepwise from the left to the right.

The algebraic approach behind stepwise tree automata yields a new systematic correspondence between queries for unranked and ranked trees. We elaborate this correspondence for monadic queries. We show that stepwise tree automata, monadic Datalog programs, and MSO can express the same monadic queries over unranked trees. We reduce this result to the case of ranked trees due to our new systematic correspondence. Specific proofs for unranked queries are not needed in contrast to [13, 7].

## 2 The algebras of unranked and ranked trees

An *unranked signature* $\Sigma$ is a set of *symbol* ranged over by $a, b$. An ordered *unranked tree* or *u-tree* $t$ over $\Sigma$ satisfies the following abstract syntax:

$$t \ ::= \ a(t_1, \ldots, t_n) \qquad \text{where } n \geq 0.$$

Unordered unranked trees were investigated in [14, 15, 18]. We identify an unranked tree $a()$ with the symbol $a$. We write $\mathsf{tree}^u$ for the set of unranked trees. The extension operator $@^u : \mathsf{tree}^u \times \mathsf{tree}^u \to \mathsf{tree}^u$ for unranked trees is depicted in Fig. 1. The extended tree $t@^u t'$ is obtained from $t$ by adjoining $t'$ as next sibling of the last child of $t$:

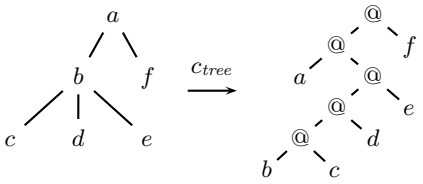
$$a(t_1, \ldots, t_n)@^u t' = a(t_1, \ldots, t_n, t')$$

Note that $a(t_1, \ldots, t_n) = a@^u t_1 @^u \ldots @^u t_n$ with parenthesis set from left to right. Tree extension $@^u$ is neither associative nor commutative.

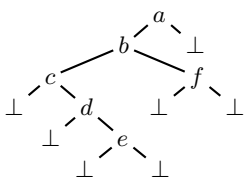
Let $\Sigma_@ = \Sigma \cup \{@\}$ be the *ranked signature* of function symbols with constants in $\Sigma$ and a single binary function symbol $@$. Ranked trees over $\Sigma_@$ are ground terms over $\Sigma_@$, i.e., binary trees that satisfy the grammar:

$$t ::= a \mid t_1 @ t_2$$

We omit parenthesis as in the $\lambda$-calculus; the ranked tree $a@b@(c@b@a)$ for instance is $(a@b)@((c@b)@a)$. We write $\mathsf{tree}^r$ for the set of ranked trees over $\Sigma_@$.




(2-a) An unranked tree $a(b(c,d,e),f)$, its ranked construction $a@(b@c@d@e)@f$.




(2-b) First-child next-sibling encoding

Ranked trees can be constructed by the binary operator $@^r : \mathsf{tree}^r \times \mathsf{tree}^r \to \mathsf{tree}^r$ which satisfies for all ranked trees $t_1, t_2$:

$$t_1 @^r t_2 = t_1 @ t_2$$

Unranked trees correspond precisely to ranked constructions with respect to the function $c_{tree} : \mathsf{tree}^u \to \mathsf{tree}^r$ which satisfies for all unranked trees $t_1, t_2$ and symbols $a \in \Sigma$:

$$c_{tree}(t_1 @^u t_2) = c_{tree}(t_1) @^r c_{tree}(t_2) \quad \text{and} \quad c_{tree}(a) = a$$

The idea of this binary construction is known from Currying. An unranked tree describes an application of a function to a list of arguments. Its binary construction represents the Curried version of this function, receiving arguments one by one. We therefore write tree extension as function application @.

The set $\mathsf{tree}^u$ of unranked trees over $\Sigma$ with the extension operation $@^u$ is a $\Sigma_@$ algebra, as well as the set $\mathsf{tree}^r$ of ranked trees over $\Sigma_@$ with the operation $@^r$.

**Proposition 1.** *The construction function $c_{tree} : \mathsf{tree}^u \to \mathsf{tree}^r$ is an isomorphism between $\Sigma_@$-algebras.*

Ranked and unranked trees thus have the same algebraic properties and the same finite automata [17, 5, 14].

Ranked constructions are binary representations of unranked trees. Previous approaches towards querying unranked trees rely on a different binary representation [8, 6], which encodes first-child and next-sibling relations. An example is given in Fig. 2-b. The new binary construction, however, permits to carry over traditional results from ranked to unranked trees more systematically.

## 3 Stepwise Tree Automata

*Stepwise tree automata* $A$ over signature $\Sigma$ are traditional tree automata ([4]) over the signature $\Sigma_@$. They consist of a finite set $\mathsf{states}(A)$ of *states*, a set

$$\mathsf{eval}^{\alpha}_{A}(a) = \{q \mid a \to q \in \mathsf{rules}(A)\}$$
$$\mathsf{eval}^{\alpha}_{A}(t_1 @^{\alpha} t_2) = \{q \mid q_1 \in \mathsf{eval}^{\alpha}_{A}(t_1),\ q_2 \in \mathsf{eval}^{\alpha}_{A}(t_2), q_1 @ q_2 \to q \in \mathsf{rules}(A)\}$$

**Fig. 3.** Ranked and unranked evaluation.

$\mathsf{final}(A) \subseteq \mathsf{states}(A)$ of *final states*, and a finite set of $\mathsf{rules}(A)$ of *transition rules* of two forms, where $a \in \Sigma$ and $q, q_1, q_2 \in \mathsf{states}(A)$:

$$a \to q \qquad \text{or} \qquad q_1 @ q_2 \to q$$

Stepwise tree automata can evaluate unranked and ranked trees, i.e. $\alpha$-trees where $\alpha \in \{u, r\}$. The *$\alpha$-evaluator* of $A$ is the function $\mathsf{eval}^{\alpha}_{A} : \mathsf{tree}^{\alpha} \to 2^{\mathsf{states}(A)}$ defined in Fig. 3. Both evaluators only differ in the interpretation of the symbol @.

**Lemma 1.** $\mathsf{eval}^{u}_{A}(t) = \mathsf{eval}^{r}_{A}(c_{tree}(t))$ *for all unranked trees $t$.*

Stepwise tree automata $A$ recognize all $\alpha$-trees $t$ that can be evaluated into a final state, i.e., $\mathsf{eval}^{\alpha}_{A}(t) \cap \mathsf{final}(A) \neq \emptyset$. The $\alpha$-language $L^{\alpha}(A)$ consists of all $\alpha$-trees recognized by $A$.

**Proposition 2.** *A stepwise tree automaton accepts an unranked tree if and only if it accepts its ranked construction, i.e., for all $A$:*

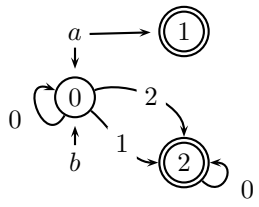
$$L^{u}(A) = c^{-1}_{tree}(L^{r}(A))$$

*Proof.* By Lemma 1 all unranked tree $t$ satisfy: $t \in L^{u}(A)$ iff $\mathsf{final}_A \cap \mathsf{eval}^{u}_{A}(t) \neq \emptyset$ iff $\mathsf{final}_A \cap \mathsf{eval}^{r}_{A}(c_{tree}(t)) \neq \emptyset$ iff $c_{tree}(t) \in L^{r}(A)$.

As a consequence, stepwise tree automata inherit numerous properties from traditional tree automata, even if interpreted over unranked trees. Recognizable unranked tree languages are closed under intersection, union, and complementation. Emptiness can be checked in linear time, and membership $t \in L^{u}(A)$ in linear time $O(|t| * |A|)$.

**Example.** We define a stepwise tree automaton $A$ with signature $\Sigma = \{a, b\}$ that recognizes all unranked trees with at least one $a$-labeled leaf.

Automaton $A$ is illustrated to the right. It has three states 0, 1, 2 two of which are final: 1, 2. A successful run of $A$ on an unranked tree assigns state 1 in a non deterministic way to a unique $a$-leaf and labels by 2 all edges visited afterwards. All other nodes and edges are labeled by 0.




a. An $a$-node can be the selected $a$-leaf: $a \to 1$.
b. Any node can be assigned to state 0: $a \to 0$, $b \to 0$, $0@0 \to 0$.
c. The edge pointing to the selected $a$-leaf may go into state 2: $0@1 \to 2$.
d. All edges visited later on may go into state 2, too: $2@0 \to 2$, $0@2 \to 2$.

In Fig. 4, we show the unique successful run of $A$ on the unranked tree $a(b, a(a, b))$ and on its construction $a@b@(a@a@b)$. Both runs bisimulate each other. They evaluate the respective tree into the final state 2.

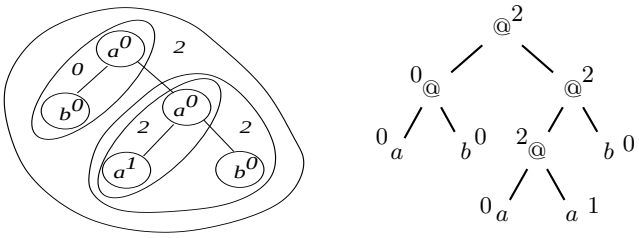



**Fig. 4.** An example run on an unranked tree and its construction

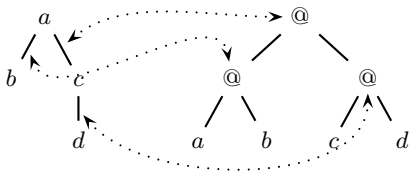



**Fig. 5.** Correspondence between edges and application nodes

## 4 Monadic Queries

Queries on unranked trees should correspond precisely to their counterparts on ranked constructions. This requires a precise correspondence between the domains of unranked trees and their ranked constructions. The domain of a ranked tree $t$ is the set of its nodes:

$$\begin{aligned} \mathsf{dom}^r(t_1 @^r t_2) &= \mathsf{dom}^r(t_1) \uplus \mathsf{dom}^r(t_2) \uplus \{\mathsf{root}\} \\ \mathsf{dom}^r(a) &= \{\mathsf{root}\} \end{aligned}$$

Disjoint union $A \uplus B$ can be implemented by $\{1\}\times A \cup \{2\}\times B$. The ranked domain of $a@(a@a)$ is then implemented by:

$$\{\mathsf{root}, (1, \mathsf{root}), (2, \mathsf{root}), (2, (1, \mathsf{root})), (2, (2, \mathsf{root}))\}$$

As usual in mathematics, we will abstract from this implementation and talk about disjoint unions as if they were simple unions.

The nodes of unranked trees correspond to leaves in ranked constructions. But what do application nodes in construction trees correspond to? The example in Fig. 5 illustrates that they correspond precisely to edges of unranked trees. Every edge was added by some application step, and vice versa, every application step adds some edge. We therefore define the domain of an unranked tree as the union of its nodes and edges.

$$\begin{aligned} \mathsf{dom}^u(t_1 @^u t_2) &= \mathsf{dom}^u(t_1) \uplus \mathsf{dom}^u(t_2) \uplus \{\mathsf{last\_edge}\} \\ \mathsf{dom}^u(a) &= \{\mathsf{root}\} \end{aligned}$$

The $\mathsf{last\_edge}$ of $t_1 @^u t_2$ links the root of $t_1$ to the root of $t_2$. Note that all nodes of $t_1 @^u t_2$ either belong to $t_1$ or $t_2$; the root of $t_1 @^u t_2$ is that of $t_1$.

$$c_{dom}(a)(\mathsf{root}) = \mathsf{root}, \quad c_{dom}(t_1@^u t_2)(\pi) = \begin{cases} \mathsf{root} & \text{if } \pi = \mathsf{last\_edge} \\ c_{dom}(t_1)(\pi) & \text{if } \pi \in \mathsf{dom}^u(t_1) \\ c_{dom}(t_2)(\pi) & \text{if } \pi \in \mathsf{dom}^u(t_2) \end{cases}$$

**Fig. 6.** Definition of the correspondence on domains $c_{dom}$.

The *correspondence* $c_{dom}(t)$ for an unranked tree $t$ is a function between the domains of $t$ and its ranked construction defined in Fig. 6 by recursion over the construction of $t$:

$$c_{dom}(t) : \mathsf{dom}^u(t) \to \mathsf{dom}^r(c_{tree}(t))$$

**Definition 1.** *A* monadic query *is a function $q$ that maps trees to subsets of their domain. This definition applies to ranked and unranked trees, i.e., for both $\alpha \in \{u, r\}$. A monadic $\alpha$-query $q$ satisfies for all $t \in \mathsf{tree}^\alpha$:*

$$q(t) \subseteq \mathsf{dom}^\alpha(t)$$

We restrict ourselves to monadic queries; more general n-ary queries map to n-tuples of elements of the domain [2]. Monadic queries over unranked and ranked trees correspond. An unranked monadic query $q$ corresponds to the ranked monadic queries $c_{query}(q)$ which satisfies for all $t \in \mathsf{tree}^r$:

$$c_{query}(q)(t) = c_{dom}(t)(q(c_{tree}^{-1}(t)))$$

All previous query notions for unranked trees [8, 6] only talk about nodes. Our extension with edges, however, is necessary to keep the symmetry to ranked queries, the reason for the simplicity of our approach.

## 5 Automata Queries

In the remainder of the paper, we will discuss regular monadic queries. These can be defined by tree automata, monadic second-order logic, and monadic Datalog. Here, we start with tree automata.

We next consider monadic queries as tree languages over the alphabet $\Sigma \times \mathsf{Bool}$. Such languages were already studied in [17]. Given a ranked tree $t$ over the signature $\Sigma \times \mathsf{Bool}$ and $i \in \{1, 2\}$, let $\mathsf{proj}^i(t)$ be the ranked tree obtained by projecting all labels in $t$ to their $i$'th component. For monadic queries $q$ and ranked trees $t$ let $\mathsf{zip}(t, q)$ be the ranked tree over the extended signature $\Sigma \times \mathsf{Bool}$ with $\mathsf{proj}^1(\mathsf{zip}(t, q)) = t$ and $\mathsf{proj}^2(\mathsf{zip}(t, q)) = q$.

**Definition 2.** *A monadic query $q$ for ranked trees is* regular *if the set $\{\mathsf{zip}(t, q) \mid t \in \mathsf{tree}^r\}$ can be recognized by a tree automaton. A monadic query $q$ for unranked trees is regular if $c_{query}(q)$ is.*

This traditional definition of regular monadic queries is simple for ranked trees but has drawbacks otherwise. First, it is not obvious how to compute such

$$\frac{a \to r(\mathsf{root}) \in \mathsf{rules}(A)}{r \in \mathsf{runs}^\alpha_A(a)} \qquad \frac{r_{|\mathsf{dom}^\alpha(t_1)} \in \mathsf{runs}^\alpha_A(t_1) \qquad r_{|\mathsf{dom}^\alpha(t_1)} \in \mathsf{runs}^\alpha_A(t_2) \quad r(\mathsf{head}^\alpha(t_1))@r(\mathsf{head}^\alpha(t_2)) \to r(\mathsf{head}(t_1@^\alpha t_2)) \in \mathsf{rules}(A)}{r \in \mathsf{runs}^\alpha_A(t_1@^\alpha t_2)}$$

**Fig. 7.** Runs $\mathsf{runs}^\alpha_A(t)$ of stepwise tree automata $A$ on $\alpha$-trees $t$.

queries efficiently, second, it is not obvious how to express them in monadic Datalog, and third, the definition of regular unranked queries depends on the correspondence to ranked queries.

Run-based queries [15, 6] with stepwise tree automata resolve these problems. We define them parametically for ranked and unranked trees. Runs of stepwise tree automaton on $\alpha$-trees $t$ associate states to all elements of the domain of $t$. Sequences of children in unranked trees are visited *stepwise* from the left to the right, while annotating edges to the children by states. See Fig. 4 for an example. More formally, let the *head* of a ranked tree be its $\mathsf{root}$; the head of a non-constant unranked tree is $\mathsf{last\_edge}$, and the head of a constant unranked tree the $\mathsf{root}$:

$$\mathsf{head}^r(t) = \mathsf{root}, \quad \mathsf{head}^u(t_1@^u t_2) = \mathsf{last\_edge}, \quad \mathsf{head}^u(a) = \mathsf{root}.$$

A *run* of a tree automaton $A$ on an $\alpha$-tree $t$ is a function labeling elements of the domain of $t$ by states of $A$:

$$r : \mathsf{dom}^\alpha(t) \to \mathsf{states}(A)$$

such that all transitions are licensed by rules of $A$. If $t = t_1@^u t_2$ then the restrictions of $r$ to the domains of $t_1$ and $t_2$ must be runs and the annotation of the head of $t$ must be justified. Furthermore, annotations of constants must be licensed. These conditions are captured by the inference rules in Fig. 7.

**Lemma 2.** *Let $A$ be a stepwise tree automaton and $t$ an $\alpha$-tree, then:*

$$\mathsf{eval}^\alpha_A(t) = \{r(\mathsf{head}^\alpha(t)) \mid r \in \mathsf{runs}^\alpha_A(t)\}$$

*Proof.* By induction on the construction of unranked trees. If $t = a$ then $\mathsf{eval}^\alpha(a) = \{q \mid a \to q \in \mathsf{rules}(A)\} = \{r(\mathsf{root}) \mid r \in \mathsf{runs}^\alpha_A(a)\}$. For $t = t_1@^\alpha t_2$ we have $\mathsf{eval}^\alpha(t) = \{q \mid q_i \in \mathsf{eval}^\alpha_A(t_i),\ q_1@q_2 \to q \in \mathsf{rules}(A)\}$ which is equal to $\{q \mid r_i \in \mathsf{runs}^\alpha_A(t_i), r_1(\mathsf{head}^\alpha(t_1))@r_2(\mathsf{head}^\alpha(t_1)) \to q \in \mathsf{rules}(A)\}$ by induction hypothesis. The definition of runs in Fig. 7 yields $\{r(\mathsf{head}^\alpha_i(t)) \mid r \in \mathsf{runs}^\alpha_A(t)\}$.

A run $r$ of an automaton $A$ on an $\alpha$-tree $t$ is *successful* if $r(\mathsf{head}^\alpha(t)) \in \mathsf{final}(A)$. Let $\mathsf{succ\_runs}^\alpha_A(t)$ be the set of successful runs of $A$ on $t \in \mathsf{tree}^\alpha$.

**Definition 3.** *A pair of a tree automaton $A$ and a set $Q \subseteq \mathsf{states}(A)$ defines a monadic query which selects all elements from the domain of a tree $t$ that are labeled by a state in $Q$ in some successful run of $A$ on $t$:*

$$\mathit{query}^\alpha_{A,Q}(t) = \{\pi \in \mathsf{dom}^\alpha(t) \mid r \in \mathsf{succ\_runs}^\alpha_A(t), r(\pi) \in Q\}$$

Selection automata [6] similarly express queries for unranked trees, but rely on universal quantification over successful runs, and use a binary encoding of unranked trees in contrast to the definition above.

**Example.** Reconsider the automaton $A$ from Section 3: $\mathsf{query}^u_{A,\{1\}}$ defines the set of all $a$-leaves in unranked trees. Note that no automaton query with a bottom-up deterministic automaton can compute the same query, since it couldn't distinguish different $a$-nodes.

**Proposition 3.** *A monadic query on ranked trees is regular if and only if it is equal to some* $\mathsf{query}^r_{A,Q}$.

The proof relies on two standard automata transformations. The idea of the transformation from regular to run based queries $\mathsf{query}^r_{A,Q}$ is memorize the Boolean values in $\mathsf{zip}(t, q)$ in automata states. In order to generalize Proposition 3 to unranked trees, we establish the correspondence between unranked and ranked run-based queries.

**Theorem 1.** *Queries with stepwise tree automata on ranked and unranked trees correspond:*

$$\mathsf{query}^r_{A,Q} = c_{query}(\mathsf{query}^u_{A,Q})$$

*Proof.* We first note that runs of stepwise automata on unranked trees and ranked constructions correspond. For all $t \in \mathsf{tree}^r$, we can prove:

$$\mathsf{runs}^u_A(c^{-1}_{tree}(t)) = \{r \circ c_{dom}(t) \mid r \in \mathsf{runs}^r_A(t)\}$$

The theorem follows from straightforward calculations. For all $t \in \mathsf{tree}^r$:

$$\begin{aligned} c_{query}(\mathsf{query}^u_{A,Q})(t) &= c_{dom}(t)(\mathsf{query}^u_{A,Q}(c^{-1}_{tree}(t))) \\ &= \{\pi \mid r \in \mathsf{runs}^u_A(c^{-1}_{tree}(t)),\ r(c_{dom}(t)^{-1}(\pi)) \in Q\} \\ &= \{\pi \mid r' \in \mathsf{runs}^r_A(t).\ r'(c_{dom}(t)(c_{dom}(t)^{-1}(\pi))) \in Q\} \\ &= \mathsf{query}^r_{A,Q}(t) \end{aligned}$$

□

## 6 Monadic Second Order Logic

We next represent regular monadic queries in ranked and unranked trees in monadic second-order logic (MSO).

The domain of the logical structure induced by an $\alpha$-tree $t$ is $\mathsf{dom}^\alpha(t)$. The signature $R^r$ for structures of ranked trees contains the follwing relation symbols:

$$R^r = \{\mathsf{child}_1, \mathsf{child}_2, \mathsf{root}, \mathsf{leaf}\} \cup \{\mathsf{label}_a \mid a \in \Sigma_@\}$$

The binary relations $\mathsf{child}_1$ and $\mathsf{child}_2$ relate nodes to their first resp. second child. Unary relations $\mathsf{label}_a$ hold for all nodes labeled by $a \in \Sigma$. Furthermore, we permit the unary relations $\mathsf{root}$ and $\mathsf{leaf}$.

Logical structures for unranked trees have the following signature:

$$R^u = \{\mathsf{first\_edge}, \mathsf{next\_edge}, \mathsf{target}, \mathsf{root}, \mathsf{leaf}\} \cup \{\mathsf{label}_a \mid a \in \Sigma_@\}$$

The binary relation first_edge holds between a node and the edge to its first child. The next_edge relation links an edge with target $\pi$ to the edge whose target is the next sibling of $\pi$. The target relation holds between an edge and its target node.

Let $x, y, z$ range over an infinite set Vars of node variables and $p, q$ over an infinite set Preds of monadic predicates. The logics $\mathsf{MSO}^\alpha$ have the following formulas:

$$\phi ::= B_n(x_1, \ldots, x_n) \mid p(x) \mid \phi \wedge \phi' \mid \neg\phi \mid \exists x\phi \mid \exists p\phi$$

where $B_n \in R^\alpha$ is a predicate with fixed tree interpretation of arity $n$. Note that the relations root and leaf could be expressed by the remaining relations in $\mathsf{MSO}^\alpha$. We add them anyway, as they will be needed in monadic Datalog later on.

Let $\phi$ be an $\mathsf{MSO}^\alpha$-formula, $t$ an $\alpha$-tree, and $\sigma$ an assignment of variables into the domain of $t$ and of predicates into the powerset of this domain. We write $t, \sigma \models_{\mathsf{MSO}^\alpha} \phi$ if $\phi$ becomes true in $t$ under $\sigma$. Every formula $\phi(x)$ with a single variable $x$ defines monadic query:

$$\mathsf{query}^\alpha_{\phi(x)}(t) \;=\; \{\sigma(x) \mid t, \sigma \models_{\mathsf{MSO}^\alpha} \phi\}$$

**Theorem 2.** *[Thatcher & Wright [17]] Monadic queries expressed in monadic second order logic over ranked trees* $\mathsf{MSO}^r$ *are regular.*

**Theorem 3.** *Ranked and unranked monadic queries expressed in monadic second-order logic correspond, and are thus regular; corresponding queries can be computed in linear time:*

$$\{\mathit{query}^r_{\phi(x)} \mid \phi \in \mathsf{MSO}^r\} = \{c_{query}(\mathit{query}^u_{\phi'(x)}) \mid \phi' \in \mathsf{MSO}^u\}$$

Fig. 8 presents forth and back translations between $\mathsf{MSO}^u$ and $\mathsf{MSO}^r$. We have to show for every $\phi \in \mathsf{MSO}^u$ that $c_{query}(\mathsf{query}^u_{\phi(x)}) = \mathsf{query}^r_{[\![\phi(x)]\!]_r}$, and the analoguous property for the back translation. We proceed by structural induction over formulas. The base cases contains the difficulty, the induction step being straightforward.

We sketch the proof for formula $\mathsf{first\_edge}(x, y)$ to illustrate the principles. Consider an $u$-tree $t$ and a variable assignment $\sigma$ under which $\mathsf{first\_edge}(x, y)$ becomes true. There exists a $u$-tree $t_0 = t_1 @ t_2$ involved in the construction of $t$ such that $\sigma(x)$ is the root of $t_1$ and $\sigma(y)$ is the edge from $t_1$ to $t_2$. Since this is the first edge, $t_1$ is a constant. Therefore, in the corresponding ranked tree, the node $c_{dom}(\sigma(x))$ is a leaf and we have $\mathsf{child}_1(c_{dom}(\sigma(y), c_{dom}(\sigma(x)))$. The converse is proved in a similar way.

The case of $\mathsf{target}(x, y)$ is more tedious as it relies on the recursive $\mathsf{lar}(x, y)$ formula, stating that $x$ is a leaf, whose last ancestor to the right is $y$. This means that $y$ denotes the up most node with $\mathsf{child}^*_1(y, x)$. A model of $\mathsf{target}(x, y)$ and its translation $[\![\phi(x)]\!]_r$ in Fig.10.

Auxiliary predicates:

$$\begin{aligned}
\mathsf{ar}'(x,p) &=_{\mathsf{def}} p(x) \wedge \forall y \forall z((\mathsf{child}_1(y,z) \wedge p(z)) \rightarrow p(y)) \\
\mathsf{ar}(x,p) &=_{\mathsf{def}} \mathsf{ar}'(x,p) \wedge \forall p'(\mathsf{ar}'(x,p') \rightarrow \mathsf{subset}(p,p')) \\
\mathsf{lar}(x,y) &=_{\mathsf{def}} \mathsf{leaf}(x) \wedge \exists p.p(y) \wedge \mathsf{ar}(x,p) \wedge (\mathsf{root}(y) \vee \exists y'\ \mathsf{child}_2(y',y))
\end{aligned}$$

Logical connectives

$$\begin{aligned}
[\![\exists x\ \psi]\!]_r &=_{\mathsf{def}} \exists x\ [\![\psi]\!]_r \\
[\![\exists p\ \psi]\!]_r &=_{\mathsf{def}} \exists p\ [\![\psi]\!]_r \\
[\![\psi \wedge \psi']\!]_r &=_{\mathsf{def}} [\![\psi]\!]_r \wedge [\![\psi']\!]_r \\
[\![\neg\psi]\!]_r &=_{\mathsf{def}} \neg[\![\psi]\!]_r \\
[\![p(x)]\!]_r &=_{\mathsf{def}} p(x)
\end{aligned}$$

Node relations

$$\begin{aligned}
[\![\mathsf{first_edge}(x,y)]\!]_r &=_{\mathsf{def}} \mathsf{child}_1(y,x) \wedge \mathsf{leaf}(x) \\
[\![\mathsf{next_edge}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{child}_1(y,x) \wedge \mathsf{child}_1(x,z)) \\
[\![\mathsf{target}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{child}_2(x,z) \wedge \mathsf{lar}(y,z)) \\
[\![\mathsf{label}_a(x)]\!]_r &=_{\mathsf{def}} \mathsf{label}_a(x) \quad (a \in \Sigma) \\
[\![\mathsf{last_edge}(x,y)]\!]_r &=_{\mathsf{def}} \exists z\ (\mathsf{lar}(x,y) \wedge \mathsf{child}_1(y,z)) \\
[\![\mathsf{root}(x)]\!]_r &=_{\mathsf{def}} \exists y\ (\mathsf{lar}(x,y) \wedge \mathsf{root}(y)) \\
[\![\mathsf{leaf}(x)]\!]_r &=_{\mathsf{def}} \exists y\ \mathsf{child}_2(y,x) \wedge \mathsf{leaf}(x)
\end{aligned}$$

**Fig. 8.** Unranked into ranked MSO

$$\begin{aligned}
[\![\exists x\ \psi]\!]_u &=_{\mathsf{def}} \exists x\ [\![\psi]\!]_u \\
[\![\exists p\ \psi]\!]_u &=_{\mathsf{def}} \exists p\ [\![\psi]\!]_u \\
[\![\psi \wedge \psi']\!]_u &=_{\mathsf{def}} [\![\psi]\!]_u \wedge [\![\psi']\!]_u \\
[\![\neg\psi]\!]_u &=_{\mathsf{def}} \neg[\![\psi]\!]_u \\
[\![p(x)]\!]_u &=_{\mathsf{def}} p(x)
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{child}_1(x,y)]\!]_u &=_{\mathsf{def}} \mathsf{first_edge}(y,x) \vee \mathsf{next_edge}(y,x) \\
[\![\mathsf{child}_2(x,y)]\!]_u &=_{\mathsf{def}} (\mathsf{target}(x,y) \wedge \mathsf{leaf}(y)) \\
&\quad \vee \exists z\ (\mathsf{target}(x,z) \wedge \mathsf{last_edge}(z,y)) \\
[\![\mathsf{label}_a(x)]\!]_u &=_{\mathsf{def}} \mathsf{label}_a(x) \quad (a \in \Sigma) \\
[\![\mathsf{root}(x)]\!]_u &=_{\mathsf{def}} (\mathsf{leaf}(x) \wedge \mathsf{root}(x)) \\
&\quad \vee \exists z\ (\mathsf{root}(z) \wedge \mathsf{last_edge}(x,z)) \\
[\![\mathsf{leaf}(x)]\!]_u &=_{\mathsf{def}} \mathsf{root}(x) \vee \exists z\ \mathsf{target}(z,x)
\end{aligned}$$

**Fig. 9.** Ranked into unranked MSO

## 7 Monadic Datalog

We next express regular monadic queries in Monadic Datalog, a logic programming language, and discuss the expressive power compared to automata and MSO queries, for ranked and unranked trees.

We consider monadic Datalog in trees without negation. The languages $\mathsf{Datalog}^\alpha$ have the same signatures as $\mathsf{MSO}^\alpha$. The programs of $\mathsf{Datalog}^\alpha$ are logic program without function symbols, predefined n-ary predicates in $R^\alpha$, and free monadic predicates $p, q \in \mathsf{Preds}$. More precisely, a program $P \in \mathsf{Datalog}^\alpha$ is a finite set of rules of the form:

$$p(x) :- \mathit{Body}$$

where *Body* is a sequence of goals with n-ary predicates $B_n \in R^\alpha$:

$$\mathit{Body} ::= B_n(x_1, \ldots, x_n) \ \mid \ p(x) \ \mid \ \mathit{Body}, \mathit{Body}'$$

Every program of $\mathsf{Datalog}^\alpha$ can be seen as a formula of $\mathsf{MSO}^\alpha$; sets of clauses are conjunctions, clauses $p(x) :- \mathit{Body}$ are universally quantified implications $\forall \mathsf{Vars}.\ p(x) \leftarrow \mathit{Body}$, and bodies *Body* conjunctions of goals.

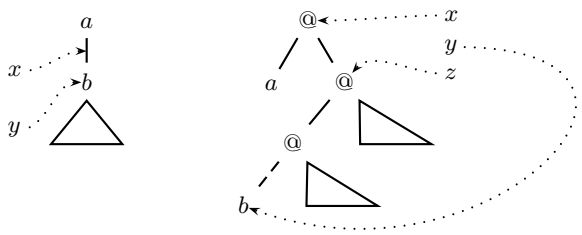



**Fig. 10.** A solution of $\mathsf{target}(x, y)$ on the left; a solution of its ranked translation $[\![\mathsf{target}(x, y)]\!]_r = \exists z\ (\mathsf{child}_2(x, z) \wedge \mathsf{lar}(z, y))$ on the right.

We interpret programs in $\mathsf{Datalog}^\alpha$ in the least fixed point semantics over the structures of $\mathsf{MSO}^\alpha$. For every program $P \in \mathsf{Datalog}^\alpha$, predicate $p \in \mathsf{Preds}$, and $t \in \mathsf{tree}^\alpha$ let $T^\omega_{P,t}(p)$ be the least solution of $P$ over the tree structure of the $\alpha$-tree $t$ for predicate $p$. This yields a notion of monadic queries:

$$\mathsf{query}^\alpha_{P(p)}(t) = T^\omega_{P,t}(p)$$

Least fixed points can be expressed in $\mathsf{MSO}$. As a consequence, every query in $\mathsf{Datalog}^\alpha$ can be expressed in linear time in $\mathsf{MSO}^\alpha$. This shows that ranked queries in $\mathsf{Datalog}^r$ are regular (Theorem 2).

Ranked monadic run-based automata queries can always be expressed in ranked monadic Datalog; the reduction is in linear time. The resulting Datalog program models the two phases of the linear time algorithm for answering automata queries: the first bottom up phase computes all states of all nodes seen in all runs of the automaton, and a top down phase selects all nodes labeled by selection states in successful runs.

**Theorem 4.** *Ranked and unranked monadic queries expressed in monadic Datalog correspond, and are thus regular:*

$$\{\mathit{query}^r_{P(p)} \mid P \in \mathsf{Datalog}^r\} = \{c_{query}(\mathit{query}^u_{P'(p)}) \mid P' \in \mathsf{Datalog}^u\}$$

*Corresponding unranked queries can be computed from ranked queries in linear time; the converse is not true.*

It suffices to encode ranked Datalog queries into corresponding unranked queries in linear time. The translation basically refines the encoding from $\mathsf{MSO}^r$ into $\mathsf{MSO}^u$: roughly, a rule $p(x) :- \mathit{Body}$ is translated into $p(x) :- [\![\mathit{Body}]\!]_u$. Conjunctions in $[\![\mathit{Body}]\!]_u$ can be replaced by commas, existential quantifications can be ommited, i.e., replaced by implicit universal quantification in rules. Disjunctions as in the definitions of $[\![\mathsf{child}_1(x, y)]\!]_u$, $[\![\mathsf{child}_2(x, y)]\!]_u$, and $[\![\mathsf{root}(x)]\!]_u$ can expressed by multiple rules. Such a rewriting, however, spoils linear time. We can circumvent this problem following Gottlob and Koch [7]: we normalise programs of $\mathsf{Datalog}^r$ into *tree marking normal form* (TMNF) in linear time before translation. TMNF programs have of forms:

$$p(x) :- B_1(x). \qquad p(x) :- q(y), B_2(y, x).$$
$$p(x) :- p_0(x), p_1(x). \qquad p(x) :- q(y), B_2(x, y).$$

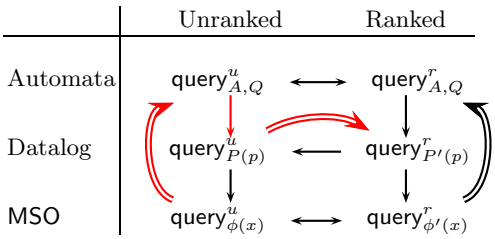



**Fig. 11.** Summary of reductions. Solid lines are in linear time, double lines are non elementary. Black lines are proved, red lines induced.

$$\frac{t = a(t_1, \ldots, t_n) \qquad \forall 1 \le i \le n : r_{|\mathsf{nodes}(t_i)} \in \mathsf{run}_H(t_i)}{a(L) \to r(\mathsf{root}(t)) \in \mathsf{rules}(H) \qquad r(\mathsf{root}(t_1)) \ldots r(\mathsf{root}(t_n)) \in L}$$
$$r \in \mathsf{run}_H(t)$$

**Fig. 12.** Runs of Hedge Automata

where $B_n$ is a n-ary predicate of $R^r$. On ranked TMNF programs the reduction $[\![\_]\!]_u$ can clearly be done in linear time.

The inverse translation can be composed from our translations so far, which are summarized in Fig. 11. We first reduce unranked Datalog queries to $\mathsf{MSO}^u$, then to $\mathsf{MSO}^r$, move to ranked automata queries and then to ranked Datalog queries. The overall reduction has nonelementary complexity.

Note that we cannot specialize the translation $[\![\_]\!]_r$ from $\mathsf{MSO}^u$ to $\mathsf{MSO}^r$ into a translation from $\mathsf{Datalog}^u$ to $\mathsf{Datalog}^r$. The problem is that we cannot express the auxilary binary predicate $\mathsf{lar}(x, y)$. Its definition is recursive, but only monadic recursive predicates can be defined in monadic Datalog.

**Corollary 1.** *Stepwise tree automata, monadic Datalog, and* MSO *capture the class of regular monadic queries over unranked trees.*

This is a corollary of our correspondences between ranked and unranked queries and traditional result on ranked trees. All unranked automata queries $\mathsf{query}^u_{A,Q}$ can be expressed in linear time in $\mathsf{Datalog}^u$, by indirection over ranked automata queries and $\mathsf{Datalog}^r$. Unranked queries in $\mathsf{MSO}^u$ can be expressed by unranked automata queries by reduction to the ranked case.

## 8 Hedge Automata

We finally show how to express monadic queries with hedge automata [16, 3] in linear time with stepwise tree automata. A hedge automaton $H$ over $\Sigma$ consists of a set $\mathsf{states}(H)$ of states, a set $\mathsf{final}(H)$ of final states, and set set of transition rules of the form $a(L) \to q$ where $L$ is a regular set of words over states.

Runs of hedge automata $H$ on unranked trees $t$ are functions $r : \mathsf{nodes}(t) \to \mathsf{states}(H)$ that satisfy the inference rule in Fig 12. A hedge automaton $H$ and a

$$\begin{aligned}
\mathsf{states}(\mathsf{step}(H)) &= \mathsf{states}(H) \uplus \biguplus_{a\in\Sigma, q\in\mathsf{states}(H)} \mathsf{states}(H_{a,q}) \\
\mathsf{rules}(\mathsf{step}(H)) &= \bigcup_{a\in\Sigma, q\in\mathsf{states}(H)} \mathsf{rules}(H_{a,q}) \\
&\quad \cup \{p \xrightarrow{\epsilon} q \mid p \in \mathsf{final}(H_{a,q}), q \in \mathsf{states}(H), a \in \Sigma\} \\
&\quad \cup \{a \to p \mid p \in \mathsf{init}(H_{a,q}), q \in \mathsf{states}(H)\} \\
\mathsf{final}(\mathsf{step}(H)) &= \mathsf{final}(H)
\end{aligned}$$

**Fig. 13.** Hedge automata into stepwise tree automata

set of states $Q \subseteq \mathsf{states}(H)$ defines a monadic query for unranked trees:

$$\mathsf{query}_{H,Q}(t) = \{\pi \in \mathsf{nodes}(t) \mid r \in \mathsf{succ\_runs}_H(t), r(\pi) \in Q\}$$

For translating hedge automata into stepwise tree automata, we need to represent all regular language $L$ in transition rules explicitly. We use a sequence of finite word automata $(H_{a,q})_{a\in\Sigma, q\in\mathsf{states}(H)}$ over the alphabet $\mathsf{states}(H)$ to do so.

**Proposition 4.** *Queries by hedge automata can be translated in linear time to queries by stepwise tree automata.*

*Proof.* Given an hedge automaton $H$ we define a stepwise tree automaton $\mathsf{step}(H)$ by unifying all subautomata $H_{a,q}$ into a single finite automaton. We then add all states of $\mathsf{states}(A)$ to this automaton, and link them to all final states $p$ of $H_{a,q}$ through $\epsilon$-transitions $p \xrightarrow{\epsilon} q$. We add rules $a \to q'$ for all initial states $q'$ of some $H_{a,q}$. The final states of $\mathsf{step}(H)$ are those in $\mathsf{final}(H)$, not those in $\mathsf{final}(H_{a,q})$. The complete construction is detailed in Fig. 13. It remains to show that every run of $H$ can be simulated by a run of $\mathsf{step}(H)$. □

**Acknowledgments:** We would like to thank our collegues from the Mostrare project for their continuous support, particularly R. Gilleron and S. Tison, and the anonymous referees for hinting us towards Currification.

## References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with Lixto. In *The Very Large Data Bases Journal*, pages 119–128, 2001.
2. A. Berlea and H. Seidl. Binary queries. In *Proceedings of Extreme Markup Languages*, Montreal, 2002.
3. A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical report, 2001.
4. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Online book, 450 pages. Available at: http://www.grappa.univ-lille3.fr/tata, 1997.
5. B. Courcelle. On recognizable sets and tree automata. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Algebraic Techniques*, volume 1, chapter 3, pages 93–126. Academic Press, 1989.
6. M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees. In *Proceedings of the IEEE Symposium on Logic In Computer Sciences*, Ottawa, 2003.

7. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. In *Proceedings of the ACM Symposium on Principle of Databases Systems*, pages 17–28, 2002.
8. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proceedings of the IEEE Symposium on Logic In Computer Sciences*, Copenhagen, 2002.
9. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPATH query evaluation. In *Proceedings of the ACM Symposium on Principle of Databases Systems*, pages 179–190. 2003.
10. G. Gottlob, C. Koch, and R. Pichler. XPATH processing in a nutshell. *ACM SIGMOD Record*, 32(2):21–27, 2003.
11. C. Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach. In *Proceedings of the International Conference on Very Large Data Bases*, 2003.
12. A. Neumann, H. Seidl. Locating matches of tree patterns in forests. In *Foundations of Software Technology and Theoretical Computer Science*, pages 134–145, 1998.
13. F. Neven and T. Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002.
14. J. Niehren and A. Podelski. Feature automata and recognizable sets of feature trees. In *Proceedings of TAPSOFT'93*, volume 668 of *LNCS*, pages 356–375, 1993.
15. H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *Proc. of the IEEE Symposium on Principles of Database Systems*, pages 155–166, 2003.
16. J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of automata theory. *J. of Comp. and Syst. Sci.*, 1:317–322, 1967.
17. J. W. Thatcher, J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Math. System Theory*, 2:57–82, 1968.
18. S. D. Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In R. Nieuwenhuis, editor, *Proceedings of RTA*, volume 2706 of *LNCS*, pages 246–263. 2003.

# Annexe D

# Conditional Random Fields for XML Trees

# Conditional Random Fields for XML Trees

Florent Jousse[1], Rémi Gilleron[2], Isabelle Tellier[2], and Marc Tommasi[2]

INRIA Futurs and Lille University, LIFL, Mostrare Project
http://www.grappa.univ-lille3.fr/mostrare
[1]jousse@grappa.univ-lille3.fr, [2]first.last@univ-lille3.fr

**Abstract.** We present XML Conditional Random Fields (XCRFs), a framework for building conditional models to label XML data. XCRFs are Conditional Random Fields over unranked trees (where every node has an unbounded number of children). The maximal cliques of the graph are triangles consisting of a node and two adjacent children. We equip XCRFs with efficient dynamic programming algorithms for inference and parameter estimation. We experiment XCRFs on tree labeling tasks for structured information extraction and schema matching. Experimental results show that labeling with XCRFs is suitable for these problems.

## 1 Introduction

We address the task of labeling XML documents with Conditional Random Fields (CRFs). Many different problems in information science, such as information extraction, data integration, data matching and schema matching, are performed on XML documents and can be dealt with using XML labeling.

Lafferty *et al* have introduced CRFs in [LMP01]. A CRF represents a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ with an associated graphical structure. CRFs have been successfully used in many sequence labeling tasks such as those arising in part-of-speech tagging [SRM04], shallow parsing [SP03], named entity recognition [ML03] and information extraction [PMWC03,SC04]; for an overview, see Sutton and McCallum's survey [SM06]. The idea of defining CRFs for tree structured data has shown up only recently. Basically, the propositions differ in the graphical structure associated with the CRFs. In [RKK+02], the output variables are independent. Other approaches such as [CC04,Sut04] define the graphical structure on rules of context-free or categorial grammars. Viola and Narasimhan in [VN05] consider discriminative context-free grammars, trying to combine the advantages of nongenerative approaches (such as CRFs) and the readability of generative ones. All these approaches apply to ranked rather than unranked trees. As far as we know, their graphical models are limited to edges.

We develop XCRFs, a new instance of CRFs that properly accounts for the inherent tree structure of XML documents. In an XML document, every node has an unlimited number of ordered children, and a possibly unbounded number of unordered attributes. The graphical structure for XCRFs is defined by: for ordered (parts of the) trees, the maximal cliques of the graph are all triangles

consisting of a node and two adjacent children; for unordered (parts of the) trees, the maximal cliques are edges consisting of a node and one child.

We define efficient dynamic programming algorithms for the inference problem and the parameter estimation problem in XCRFs. Because of the unranked property of XML trees, algorithms for XCRFs implement two recursions: an horizontal recursion following the child ordering and a vertical recursion following the sibling ordering.

We have implemented XCRFs in a system for labeling XML trees (`treecrf.gforge.inria.fr`). The system allows to label elements, attributes and text nodes of XML trees. In a first set of experiments, we show that attribute features and triangle features significantly improve the performance of XCRFs in XML tree labeling tasks. To the best of our knowledge, no alternative system for labeling trees exists, because so far only *ad hoc* solutions have been used. Nevertheless, in a second set of experiments, we have applied XCRFs on XML tree labeling tasks for schema matching and structured information extraction. For instance, for schema matching on the real estate domain, we show that the XCRF system performs really well although it does not use domain constraints or integrity constraints like the LSD system [DDH01].

## 2 Conditional Random Fields

We refer to [SM06] for a complete introduction to CRFs. A CRF is a conditional distribution with an associated graphical structure. Let $\boldsymbol{X}$ and $\boldsymbol{Y}$ be two random fields, let $\mathcal{G}$ be an undirected graph over $\boldsymbol{Y}$. Let $\mathcal{C}$ be the set of all cliques of $\mathcal{G}$. The conditional probability distribution is of the form:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{c \in \mathcal{C}} \psi_c(\boldsymbol{y}_c, \boldsymbol{x})$$

where $\psi_c$ is the potential function of the clique $c$ and $Z(\boldsymbol{x})$ is a normalization factor. Each potential function has the form:

$$\psi_c(\boldsymbol{y}_c, \boldsymbol{x}) = \exp\Big(\sum_k \lambda_k f_k(\boldsymbol{y}_c, \boldsymbol{x}, c)\Big)$$

for some real-valued parameter vector $\Lambda = \{\lambda_k\}$, and for some set of real-valued feature functions $\{f_k\}$. This form ensures that the family of distributions parameterized by $\Lambda$ is an exponential family. The feature function values only depend on $\boldsymbol{y}_c$, *i.e.* the assignments of the random variables in the clique $c$, and the whole observable $\boldsymbol{x}$. The two main problems that arise for CRFs are:

**Inference:** given an observable $\boldsymbol{x}$, find the most likely labeling $\hat{\boldsymbol{y}}$ for $\boldsymbol{x}$, *i.e.* compute $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|\boldsymbol{x})$.

**Training:** given a sample set $S$ of pairs $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}$, learn the best real-valued parameter vector $\Lambda$ according to some criteria. In this paper, the criterion for training is the maximum conditional penalized log-likelihood.

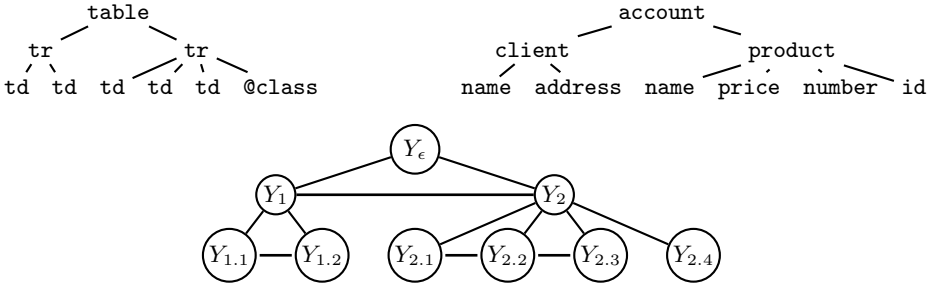



**Fig. 1.** An ordered unranked tree, its labeling and its graph.

## 3 CRFs for XML Trees

XML documents are represented by their DOM tree. We only consider element nodes, attribute nodes and text nodes of the DOM representation. Other types of nodes[1] are not concerned by labeling tasks. Attribute nodes are unordered, while element nodes and text nodes are ordered. We identify a node by a position which is a sequence of integers $n$ and we denote by $x_n$ the symbol in position $n$ in the tree $\boldsymbol{x}$. The $k$ ordered children of a node in position $n$ are identified by positions $n.1$ to $n.k$. The unordered children are identified by positions $n.(k+1)$ and higher. As a running example, consider the two XML trees $\boldsymbol{x}$ (on the left) and $\boldsymbol{y}$ (on the right) in Figure 1. The set of nodes for both trees is $\{\epsilon, 1, 1.1, 1.2, 2, 2.1, 2.2, 2.3, 2.4\}$, $\epsilon$ being the root. The symbol in position 2.1 in $\boldsymbol{x}$ is `td`; in its labeling $\boldsymbol{y}$, the label in position 2.1 is `name`. $\{2.1, 2.2, 2.3, 2.4\}$ is the set of children of 2, where 2.1, 2.2 and 2.3 are ordered children and 2.4 is an unordered child of 2.

With every set of nodes, we associate a random field $\boldsymbol{X}$ of observable variables $X_n$ and a random field $\boldsymbol{Y}$ of output variables $Y_n$ where $n$ is a position. The realizations of $X_n$ will be the symbols of the input trees, and the realizations of $Y_n$ will be the labels of their labelings. In the following, we freely identify realizations of these random fields with ordered unranked trees.

For their ordered parts, the structure of XML trees is governed by the sibling and the child orderings. We translate this structural property into XCRFs, by defining triangle feature functions that have the form:

$$f_k\big(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i\big) \ . \tag{3.1}$$

Their arguments are the labels assigned to the node $n$ and to two consecutive children of it ($n.i$ and $n.(i+1)$), the whole observable $\boldsymbol{x}$, and the identifier of the clique in the tree $n.i$. For unordered parts of XML trees there is no next-sibling ordering, feature functions are thus only defined over nodes (node features) and pairs of nodes (edge features).

In Figure 1 (bottom), we show the graph for our running example. We denote by $\mathcal{C}$ the set of cliques in the dependency graph. We use $f_k$ to index every

[1] comments, processing instructions...

feature function. To shorten the presentation, node, edge and triangle feature are all written like in (3.1). Each $f_k$ is associated with a real-valued parameter $\lambda_k$, defining the vector $\Lambda = \{\lambda_k\}$. It is worth pointing out that the same set of feature functions with the same parameters is used for every clique in the graph. The conditional probability distribution for an XCRF can be written as:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{n.i \in \mathcal{C}} \exp\big(\sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i)\big)$$

$$\text{where } Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \Big( \prod_{n.i \in \mathcal{C}} \exp\big(\sum_k \lambda_k f_k(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i)\big)\Big)$$

**Inference** XCRFs are a particular case of graphical models. The treewidth of undirected graphs for XCRFs is 2. For every graph associated with an XCRF, a junction tree can be computed in linear time. Then the belief propagation algorithm can be applied [TAK02,SRM04]. However using the knowledge of the tree-shaped graphical structures associated with XCRFs, we propose a dynamic programming algorithm for inference in XCRFs. The algorithm is based on the dynamic programming algorithm for probabilistic context-free grammars but introduces another set of variables due to the possibily large number of children.

**Training** Training an XCRF means learning its parameter vector $\Lambda$. We are given iid training data $S$ of pairs of the form (observable tree, labeled tree). Parameter estimation is performed by penalized maximum likelihood. The conditional log-likelihood, defined as $\mathcal{L}_\Lambda = \sum_{(\boldsymbol{x},\boldsymbol{y}) \in S} \log p(\boldsymbol{y}|\boldsymbol{x}; \Lambda)$, is used. This function is concave and the global optimum is the vector of parameters with which the first derivative is null. However, finding analytically this derivative with respect to all the model parameters is impossible. The L-BFGS gradient ascent [BLNZ95], which requires the computation of the partial derivatives of $\mathcal{L}_\Lambda$ for each parameter, is therefore used. To make these computations tractable, we introduce a dynamic programming algorithm using both forward-backward variables and inside-outside variables.

$Z(\boldsymbol{x})$ can be computed in $O(N \times M^3)$ where $N$ is the number of nodes of $\boldsymbol{x}$ and $M$ is the number of distinct labels in $\mathcal{Y}$. This result can be extended to the computation of the marginal probabilities in the gradient. This leads to an overall complexity for training in $O(N \times M^3 \times G)$ where $N$ is the total number of nodes of the trees in the input sample $S$, $M$ is the number of distinct labels in $\mathcal{Y}$, and $G$ is the number of gradient steps. For linear chain CRFs only a factor $M^2$ occurs.

## 4 Experiments with the XCRF System

### 4.1 The XCRF System

The XCRF model is implemented by a freely available JAVA library[2]. For training, the parameters are estimated by maximizing the penalized log-likelihood. This implementation is a stochastic system which allows to label element, attribute and text nodes of XML trees. It provides the ability to not label some nodes by assigning them a label which means "not labeled". Labeling a medium-sized (about 1000 nodes) XML tree with an XCRF built on 100 features is almost immediate. An XCRF is specified by an XML file. Feature functions are 0-1 valued functions defined by XPATH expressions. There are node features, edge features, attribute features (edge features for unordered children) and triangle features.

### 4.2 Feature Generation

Users can easily introduce domain knowledge via the definition of feature functions in XCRFs. But to be fair, in all our experiments, feature functions are automatically generated from the training set, using a syntactic and domain-independent procedure. **Structure features** are node features, edge features and triangle features which are based on node symbols and labels. For instance, let $1_p$ be 1 if and only if $p$ is true. Consider a tree $\boldsymbol{x} = \texttt{tr}(\texttt{td},\texttt{td})$ and its labeling $\boldsymbol{y} = 0(1,2)$. The triangle feature $f(y_n, y_{n.i}, y_{n.(i+1)}, \boldsymbol{x}, n.i) = 1_{\{y_n=0\}}\ 1_{\{y_{n.i}=1\}}\ 1_{\{y_{n.(i+1)}=2\}}\ 1_{\{x_n=tr\}}\ 1_{\{x_{n.i}=td\}}\ 1_{\{x_{n.(i+1)}=td\}}$ is generated together with two edge features and three node features. **Attribute features** are based on attribute values. We also preprocess documents: additional information on the structure (number of children, depth, etc.) or on the textual content of a leaf (ContainsComma, ContainsColon, IsANumber, etc) are encoded into attributes. Attribute features are therefore generated from both original and preprocessed attributes. Consider a node $n$ of $x$ which is labeled with 0 and has an attribute $a$ at position $i$ whose value is 2 labeled with 1. An attribute feature $f(y_n, y_{n.i}, \boldsymbol{x}, n.i) = 1_{\{y_n=0\}}\ 1_{\{y_{n.i}=1\}}\ 1_{\{x_{n.i}=2\}}$ is generated.

### 4.3 Interest of Triangle Features

We first want to experimentally evaluate the significance of the triangle features, and to show that learning is successful with small datasets.

To evaluate our system we provide recall, precision and F1-measure over the number of nodes. Precision and recall are the ratio of the number of correctly labeled nodes to respectively the total number of labeled nodes and the total number of nodes which had to be labeled. Using a simple accuracy measure would take into account the nodes which are not labeled. The results would therefore be biased.

**Triangle features.** The "Courses" dataset[4] collected by Doan consists of 960 XML documents of about 20 nodes each, containing course information from five

[2] `http://treecrf.gforge.inria.fr/`

| Features | F1 | %docs |
|---|---|---|
| Edge | 52.19 | 0 |
| Edge & Attribute | 84.91 | 26.51 |
| Triangle | 93.62 | 62.24 |
| Triangle & Attribute | 99.84 | 98.93 |

| Nb Docs | F1 | %docs |
|---|---|---|
| 5 | 82.15 | 50.97 |
| 10 | 91.34 | 69.14 |
| 20 | 96.65 | 82 |
| 50 | 98.77 | 93.82 |

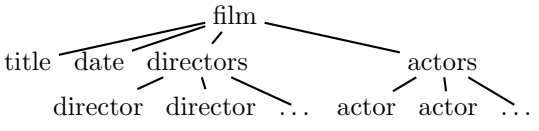
**Table 1. Left:** Triangle features versus edge features. **Right:** Varying the size of the training set

universities. All XML tags are removed and replaced with a unique and therefore uninformative one. The task consists in relabeling the XML documents with the original 14 distinct tags. We train XCRFs with node and edge structure features then with node, edge and triangle features. In both cases, we also train the XCRFs with or without attribute features. Results of Table 1 are means of 5 iterations in which XCRFs are trained on one fifth of the corpus and evaluated on the other 4 fifths. They show the F1-measure on document nodes and the percentage of XML documents that were completely correctly labeled. They confirm the relevance of triangle features. Indeed, results are far better with triangle features than when attribute features (containing also structural information) are added to edge features. Labeling is almost perfect with triangle and attribute features.

**Number of examples.** We consider the same task using structure features and attribute features. This time, we want to evaluate the impact of the number of examples in the learning set. Experimental results in Table 1 show that with only 20 documents for learning (192 in the previous experiment), the system already achieves more than 96% in F1-measure. When using more documents, the performances still rise, but very slowly. It is also interesting to note that when the XCRF has about 300 feature functions, the labeling of these documents is still immediate.

### 4.4 Experiments on Applicative Domains

Now, we apply our tree-labeling learning system to two applicative domains. The first one is structured information extraction, the second one is schema matching, both considered as a labeling task.




**Structured Information Extraction.** For this experiment, we take 558 XML documents from the MovieDB corpus used for the XML Document Mining Challenge[5]. The average number of nodes in these documents is 170. The task is to extract data according to the target DTD shown on the right. This purpose can be achieved by labeling the input XML documents according to this DTD. Note that two nodes with the same input tag might be labeled with different output tags, depending on the context. For instance, the tag `name` can be labeled either by `actor` or by `director`. To evaluate the XCRF

[4] `http://anhai.cs.uiuc.edu/archive`

| Nb docs | Recall | Precision | F1 | % docs |
|---|---|---|---|---|
| 5 | 100 | 97.91 | 98.94 | 71.32 |
| 10 | 100 | 99.55 | 99.77 | 89.84 |
| 20 | 100 | 99.99 | 99.99 | 99.63 |

**Table 2.** Results on MovieDB Structured IE.

system on this task, we run 10 experiments, each time randomly choosing 5, 10 or 20 labeled documents on which an XCRF is trained and tested on the remaining documents. The results, very promising, are provided in Table 2.

**Schema Matching.** For the problem of schema matching, we evaluate XCRFs on the "Real Estate I" dataset[4], collected by Doan. This corpus, built from five real estate websites, describes house listing information and contains about 10000 documents of about 35 nodes each. Each of the five sources has its own schema. A unique mediated schema with 16 tags is also known. The task thus consists in labeling the nodes of the documents in their source schema with their corresponding tag in the mediated schema. As in [DDH01], for every experiment, it is supposed that mappings are known for three sources out of the five ones and that documents labeled according to the mediated schema are the training set. We run 20 experiments, each time choosing at random 3 sources with which an XCRF with structure and attribute features is learnt. We took 5 labeled documents from each source. All the documents from the remaining 2 sources are used to evaluate the XCRF. The system achieved an excellent recall of 99%, and 88% of F1-measure. Unfortunately, these results can not be compared to the ones achieved by the LSD system given in [DDH01]. Indeed, [DDH01] measure the ratio of correct mappings between a tag in the source schema and its corresponding tag in the mediated schema. Since we are using a conditional model to label tree nodes, the same tag in the source schema can be mapped to different tags in the mediated schema depending on the context. Therefore, we can not provide this measure. However, it is still worth noting that we achieve very good results without using the domain knowledge, such as domain constraints or integrity constraints, that was used in the LSD system.

## 5 Conclusion

We have shown that XCRFs are a very relevant model for labeling XML trees. Experimental results show the significance of attribute and triangle features. Also, preliminary experimental results show that XCRFs perform very well on tasks such as structured information extraction or schema matching. Various extensions are being considered.

First, when labeling an XML tree, one can be interested in labeling inside the text nodes, *i.e.* assigning different labels to different parts of text. To do so, the use of both the tree structure and the text sequence is needed. Thus, combining

[5] `http://xmlmining.lip6.fr/Corpus`

linear chain CRFs and XCRFs could be a good way of taking advantage of both the structured and the linear view of XML documents.

Second, in the XCRF system, parameter estimation is done by maximizing the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, meaning we try to maximize the number of completely correctly labeled XML documents. Sometimes, for instance in schema matching, one might instead prefer to maximize the number of correctly labeled nodes. To do so, another criterion for learning could be the maximum pseudo-likelihood, which consists in maximizing the marginal probabilities $p(y_n|\boldsymbol{x})$.

## References


[BLNZ95] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Computing*, 16(5):1190–1208, 1995.

[CC04] Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proc. of ACL*, pages 103–110, 2004.

[DDH01] AhHai Doan, Pedro Domingos, and Alon Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proc. of the ACM SIGMOD Conference*, pages 509–520, 2001.

[LMP01] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pages 282–289, 2001.

[ML03] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields. In *Proc. of CoNLL'2003*, 2003.

[PMWC03] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *Proc. of the ACM SIGIR*, pages 235–242, 2003.

[RKK+02] S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *Proc. of ACL*, pages 271–278, 2002.

[SC04] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Proc. of NIPS*, pages 1185–1192, 2004.

[SM06] Charles Sutton and Andrew McCallum. *Introduction to Statistical Relational Learning*, chapter An Introduction to Conditional Random Fields for Relational Learning. MIT Press, lise getoor and ben taskar edition, 2006.

[SP03] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proc. of HLT-NAACL*, pages 213–220, 2003.

[SRM04] Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proc. of ICML*, pages 783–790, 2004.

[Sut04] Charles Sutton. Conditional probabilistic context-free grammars. Master's thesis, University of Massachusetts, 2004.

[TAK02] Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proc. of UAI02*, pages 485–492, 2002.

[VN05] Paul Viola and Mukund Narasimhan. Learning to extract information from semistructured text using a discriminative context free grammar. In *Proc. of the ACM SIGIR*, pages 330–337, 2005.

[Wal02] Hanna Wallach. Efficient training of conditional random fields. Master's thesis, University of Edinburgh, 2002.

# Annexe E

# Interactive Tuples Extraction from Semi-Structured Data

# Interactive Tuples Extraction from Semi-Structured Data

Rémi Gilleron Patrick Marty Marc Tommasi
Fabien Torre
INRIA Futurs and Lille University; email: first.last@univ-lille3.fr

## Abstract

*This paper studies from a machine learning viewpoint the problem of extracting tuples of a target $n$-ary relation from tree structured data like XML or XHTML documents. Our system can extract, without any post-processing, tuples for all data structures including nested, rotated and cross tables. The wrapper induction algorithm we propose is based on two main ideas. It is incremental: partial tuples are extracted by increasing length. It is based on a representation-enrichment procedure: partial tuples of length $i$ are encoded with the knowledge of extracted tuples of length $i-1$. The algorithm is then set in a friendly interactive wrapper induction system for Web documents. We evaluate our system on several information extraction tasks over corporate Web sites. It achieves state-of-the-art results on simple data structures and succeeds on complex data structures where previous approaches fail. Experiments also show that our interactive framework significantly reduces the number of user interactions needed to build a wrapper.*

## 1 Introduction

In less than ten years, XML has become a standard for semi-structured data exchange and storage. From its preliminary usage in Internet technologies, XML influence has grown to become widely used now for corporate data[1]. XML usage includes document-like data produced for instance by word processors and spreadsheets, data for software interoperability (UDDI, WSDL). In the meantime, with the generalization of decision support systems, high valued corporate data such as economic statistics are generated by reporting tools[2] and are published as semi-structured documents. All such data participates in a new source of information that end-users want to query in an easy way.

But end-user accessibility of XML or XHTML data, by means of query tools, is more problematic than in the classical and long-standing experienced case of relational databases. This situation is due either to the lack of explicit and coercive data types or due to the extensibility nature of XML that allows great heterogeneity of data.

Therefore, in spite of standardized query languages like XPATH, a common attitude is to develop specific programs for each query over semi-structured data. Recently, techniques have appeared to assist end-users in designing such programs. For instance, the Lixto system [2] allows to define queries or so-called *wrappers* over semi-structured data in a visual and interactive framework. Unfortunately, specifying wrappers with such systems can be too difficult for end-users. Alternate or complementary approaches relying on machine learning techniques have been considered. As in a query by example system, an end-user annotates data to extract and the system induces an appropriate wrapper. Machine learning approaches have been widely developed for Web information extraction [3, 5, 9, 11, 14, 16, 17]. These systems are limited so far to simple Web pages containing lists of results *à la Google* or tables.

But data organizations in tree structured documents may be intricate. For instance, let us consider as a running example a (part of a) page of the BEA Web site from www.bea.gov in Fig. 1. The corresponding HTML tree is given in Fig. 2. Let us consider the target relation (Country, Year, Exports, Balance). The tuples (France, 1986, 10.130, 7.119) and (France, 1987, 11.701, 7.947) are intertwined in the HTML tree. Thus extracting tuples in the document order should fail. Also it could be noted that values for the component Country are factorized over several tuples.

Our objective is a friendly system for extracting tuples from XML or XHTML documents whatever the structure used to store tuples is. We follow the wrapper induction approach which is licensed because the generation process implies regularities of semi-structured data. We first study, in Section 2, different ways to encode $n$-ary relations in XML, that is the manner to store tuples in XML trees. We define five base cases: lists, tables, rotated tables, nested tables and cross tables. Related work is described in Section 3. It is shown that existing systems for extracting tuples from Web documents only deal with lists and tables. Other systems

[1] See for instance www.oasis-open.org
[2] See for instance jasperreports.sourceforge.net

**Table 12. U.S. International Transactions, by Selected Countries (published annually) - France** [16]

[Millions of dollars]

| Line | (Credits +; debits -) [1] | 1986 | 1987 |
|---|---|---|---|
| | **Current account** | | |
| 1 | **Exports of goods and services and income receipts** | | |
| 2 | Exports of goods and services | 10,130 | 11,701 |
| 3 | Goods, balance of payments basis[2] | 7,119 | 7,947 |

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

**Table 12. U.S. International Transactions, by Selected Countries (published annually) - Germany** [16]

[Millions of dollars]

| Line | (Credits +; debits -) [1] | 1986 | 1987 |
|---|---|---|---|
| | **Current account** | | |
| 1 | **Exports of goods and services and income receipts** | | |
| 2 | Exports of goods and services | 14,738 | 17,184 |
| 3 | Goods, balance of payments basis[2] | 10,461 | 11,525 |

Today's Date: February 1, 2006 Release Date: December 16, 2005 Next Release Date: March 14, 2006
Earliest Year Revised on December 16, 2005: No Revision

**Figure 1. Part of a sample page of BEA web site.**

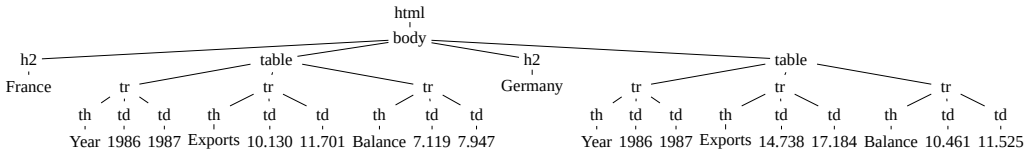



**Figure 2. Simplified HTML tree for a page of the BEA Web site (Fig. 1). It corresponds to rotated tables with pivot values**

rely on unary wrappers and a post processing to enable tuple extraction. But let us consider the BEA example and the target relation (Country, Year, Exports, Balance) and let us suppose that the sets of values for each component of the target relation are known, the post processing needs the knowledge of the tree structure depicted in Fig. 2. Writing such a post processing procedure is not easy for an end-user. Moreover, writing by hand such a procedure is in conflict with the example-based approach.

Thus a machine learning approach for extracting tuples should allow to extract tuples for all data organizations. The number of tuples in a tree is exponential in the length of tuples. Therefore, direct approaches fail and we propose a system based on two main ideas. It is *incremental*: partial tuples are extracted by increasing length. It is based on a *representation-enrichment procedure*: partial tuples of length $i$ are encoded with the knowledge of extracted tuples of length $i-1$. In Section 4, we present the extraction algorithm. At each step of the incremental process, a binary classification procedure is used. The attribute-value representation schemas use both the `DOM` view (tree structure) and the `yield` view (textual contents) of `XML` data. The induction procedure from completely annotated documents, in which all tuples to be extracted are annotated, is given in Section 5. At each step $i$ of the incremental process, a classifier is learned from examples. Positive examples are encodings of partial tuples which are projections of tuples to be extracted. Negative examples are encodings of partial tuples not to be extracted while the $i-1$-th partial tuple is to be extracted. This ensures that the number of examples is linear in the number of annotated tuples.

The system suffers of one main flaw: completely annotated examples are required. As our objective is to build a friendly end-user system for wrapper generation with machine learning techniques, it is mandatory to be able to build accurate wrappers in a minimal amount of time. In Section 6, we integrate our approach in an interactive framework in order to reduce the number of user actions. This implies to modify the algorithms to learn from partially annotated examples, in which some tuples to be extracted are annotated as positive examples and some tuples not to be extracted are annotated as negative.

In Section 7, we present experimental results of our prototype. It has been evaluated on standard RISE [18] data sets and corporate Web sites in the domains of statistics, meteorology and economics. These data sets cover the simplest organizations (lists and tables) and more complex ones (rotated tables, nested tables and cross tables). First,

we consider the case of completely annotated examples. Our system achieves excellent results without any post-processing. On simple datasets it reaches state-of-the-art results and it succeeds on intricate ones where other systems fail. Second, we study the behaviour of our algorithm in an interactive framework. We show that this interactive approach significantly improves the time needed to build wrappers when the number of tuples in each document is large. It also reduces the quantity of information provided by the user to define accurate wrappers.

## 2 Tuples in XML data

We discuss how tuples can be embedded in XML data. Our case study was done for XHTML data from different Web sites. We present the five base cases:

***Case 1.*** Tuples can be stored consecutively in the same order. For instance consider a list of search results like Google or eBay. Tuples do not overlap in the XHTML document. Two cases should be distinguished. In the first one, the tree structure of the XHTML document is poor and the problem is equivalent to a problem for sequences. In the second one, the list constructor of XHTML is used. In this case, each tuple is in a different item. The case of nested lists is quite similar to the case of tables we present now.

***Case 2.*** A second structure is a table with component names in the first row and data in the following rows[3]. Tuples again do not overlap in the XHTML document. In the DOM view the least common ancestors of each tuple are distinct. So it is easy to distinguish between tuples using the tree structure or the XHTML tags in the document. Nested lists lead to similar properties.

***Case 3.*** For relations with more components than data instances, the previous organization is often rotated. Slot names are in the first column and data are in the following columns[4]. Another example is given by the tables from www.bea.gov in Fig. 1 for the target relation (Year, Exports, Balance) for France. It is shown in Fig. 2 that tuples (1986, 10.130,7.119) and (1987, 11.701, 7.947) are intertwined. It should also be noted that different components of a tuple occur at the same position in the DOM tree. For instance, 1987, 11.701 and 7.947 are below the third td of the tr tags of the table.

***Case 4.*** A nested structure is used to avoid repetitions and to shorten the data presentation. This is common when issued from reporting tools. Consider for instance the address report from JASPERREPORTS [5] where the city name is factorized among several citizens records. Another example is given by the tables from www.bea.gov in Fig. 1 for the target relation (Country, Year, Exports, Balance). The country name is factorized and it is shown in Fig. 2 that component values occur at different depths in the DOM tree.

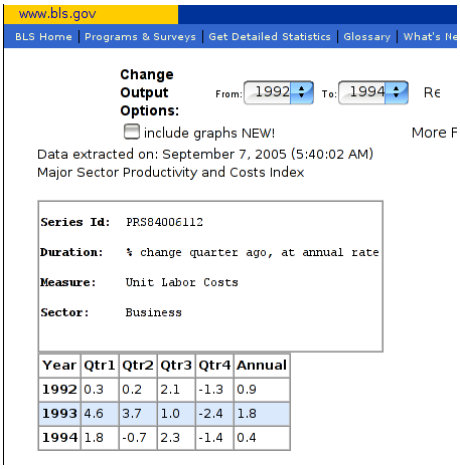



**Figure 3. Part of a sample page of BLS web site.**

***Case 5.*** Cross-tables are used to present multi-dimensional relations. This is frequent for statistics and reports. This case shares properties with the two preceding cases: a subset of the component values are stored as in the case 3 but the rest is stored according to case 4. An example from BLS www.bls.gov is given in Fig. 3. The DOM tree is given in Fig. 4. Let us consider for instance the target relation (Year, Quarter, Unit Labor Costs). Examples of tuples are (1992, Qtr1, 0.3) and (1993, Qtr3, 1.0).

For the two first cases, tuples are stored consecutively while, for the three last cases, tuples are intertwined both in the sequential view and in the DOM view. For the three last cases, it is necessary to count in order to construct tuples because component values occur at the same position in different subtrees. It should be noted that variants are frequent in XML or XHTML data. They consist in mixing several possible layouts, merging some leaves of the tree or placing several relations in the same structure. Also, building tuples can be more intricate when there are missing values.

## 3 Related work

We relate to [12] for a survey on Web data extraction tools. Here we focus on wrapper induction tools for Web documents. We discuss the supervised and unsupervised approaches.

[3] see the table of market indices on quotes.nasdaq.com/aspx/marketindices.aspx

[4] see the table of http://www.bls.gov/eag/eag.IA.htm

[5] jasperreports.sourceforge.net/samples/DataSourceReport.html

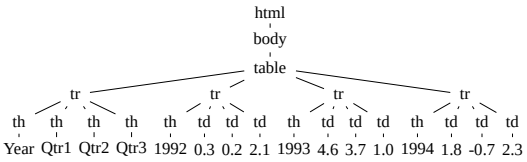



**Figure 4. Tree organization for a cross-table from BLS Web site presented in Fig. 3 restricted to the three first quarters.**

The supervised wrapper induction problem has been addressed by many authors. Several systems have emerged from this active research field, working from manually annotated documents. Let us cite WIEN [11], SOFT MEALY [9], BWI [7], STALKER [14], SQUIRREL [3], among others. They achieve very good performance on unary extraction tasks. Wrapper induction approaches for $n$-ary tuples extraction include Kushmerick seminal works [11] with LR wrappers and transducers induction in [9]. More recently Thomas proposes an approach based on Inductive Logic Programming [17]. These systems are designed for `HTML` documents generated from a back-end database, such as commercial sites and online shops. For instance, LR wrappers work when tuples are stored consecutively in the sequential view of `HTML` documents. Thus, to the best of our knowledge, wrapper induction systems for $n$-ary relations are able to deal with the most simple cases: lists, nested lists and tables.

For $n$-ary extraction, it is often argued that a post-processing that builds tuples from components extracted by $n$ unary wrappers is sufficient to solve the $n$-ary extraction problem. Indeed, this post processing is easy for cases 1 or 2 when there are no missing values. With missing values, user interactions are required to label data or to correctly re-associate tuple components [5, 10]. We argue that this is unsatisfactory for at least two reasons. First, we have shown in the introduction that, even with perfect unary wrappers, the post-processing procedure may be complex. Second, learning-based solutions have been promoted to avoid writing wrappers by hand. Thus it is not fair to ask the end user to write programs for constructing tuples from the outputs of several unary wrappers.

The unsupervised approach is automatically trying to avoid manual labeling. Unsupervised learning methods are based on document alignment [1, 4, 13, 20] or grammatical inference techniques [6]. However, as indicated by Zhai and Liu in [19], these automatic methods are less accurate than the systems that ask the user to label training pages. Manual post-processing is needed for the user to identify what he/she is interested in. For instance, consider the tables from www.bea.gov in Fig. 1, the system presented in [20] identifies the two data regions corresponding to the two tables. But if we consider the target relation (Country, Year, Exports, Balance), the problem of extracting tuples is still present. Moreover, in case 4, factorized values may not be found by an unsupervised system. Therefore, extracting tuples of records from the output of automatic systems can be as difficult as wrapping original documents. Nevertheless, automatic systems could be a useful preprocessing for large documents with few data records in order to identify the data regions.

To sum-up, existing systems based on machine learning techniques solve simple data organizations (Cases 1 and 2), but fail on complex data organizations (Cases 3-5).

## 4 The extraction process

We present the incremental extraction procedure which is based on data enrichment. The first component of the target $n$-ary relation is called the *seed*. The seed extraction is cast as a binary classification problem, which consists in deciding whether a leaf is to be extracted or not. Figure 5 briefly describes the 56 attributes used by $rep_1$ to encode a leaf. The reader should note that attributes are defined from several views over `XML` or `XHTML` documents: nodes properties over the *`DOM` view*; textual values over the *`yield` view* which is the concatenation of text leaves' contents obtained with a depth-first left-right exploration of the `DOM` view.

Extraction of partial tuples of length $i$ ($i \neq 1$) is cast as a binary classification problem, which consists in deciding whether a tuple of length $i$ should be extracted or not. Tuples of length $i$ are encoded in an attribute-value representation denoted by $rep_i$. Let us consider a tuple $(l_0, \ldots, l_{i-1}, l_i)$, $rep_i$ includes an encoding of dependencies between $l_i$ and the partial tuple $(l_0, \ldots, l_{i-1})$. For encoding dependencies, we use node couples attributes which are presented in Fig. 6. Again node couples attributes are defined over the `DOM` view and the `yield` view. In order to have a fixed number of attributes whatever the length of the partial tuple is, $rep_i$ encodes a partial tuple $(l_0, \ldots, l_{i-1}, l_i)$ by the leaf representation of $l_i$, node couples attributes for

**Node representation** label, position in father's children sequence, depth and height, number of children, size of the subtree under the node, the label of its previous and next siblings, the value of the `class` `XHTML` attribute. There are 9 attributes.
**Leaf representation** node representation of the leaf plus node representation of its 5 ancestors in the `DOM` view plus the previous and next leaves' contents (without tokenisation) in the `yield` view. There are 56 attributes.

**Figure 5. Attributes for leaf representation.**

**node couples attributes**: for a pair of nodes $(p, m)$ attributes are: attributes for the node representation of the nearest common ancestor $a$ between $p$ and $m$; the length of the shortest path between $p$ and $a$, between $p$ and $m$ and between $a$ and $m$ in the `DOM` view and in the binary tree encoding of the `DOM` view; the number of textual leaves between $p$ and $m$ in the `yield` view; and for each $1 \leq k \leq 5$; the difference between the position of the $k^{th}$ ancestor of $p$ and of the $k^{th}$ ancestor of $m$ in their father's children sequence. There are 21 attributes for each pair of nodes.

**Figure 6. Attributes for pairs of nodes**

the pair $(l_0, l_i)$ (dependency with the seed) and node couples attributes for the pair $(l_{i-1}, l_i)$ (dependency with the previous component). There are 98 attributes for a tuple representation and they are all independent of $i$.

The extraction algorithm for a target $n$-ary relation is the algorithm 1. It takes as input a document $d$ and $n$ classifiers $c_1, \ldots, c_n$ according to the representation schemas $rep_1, \ldots, rep_n$. First, tuples of length 1 are extracted. Then, for each $i$, in line 4 a set $S_i$ of candidate tuples to be extracted is defined by adding a leaf to every extracted tuple of length $i-1$. Tuples of length $i$ are extracted in line 5: the representation procedure $rep_i$ issues a record; records are then given to a classification procedure $c_i$ that labels each of them as positive or negative. Every tuple of length $i$ classified as positive is extracted. This allows us to handle factorized values. Indeed, when one of the $i-1$ components is a factorized values, several tuples of length $i$, constructed with the same tuple $t_{i-1}$ of length $i-1$, must be extracted. Line 6 allows us to handle missing values. When no tuple $t_i$, constructed from some $t_{i-1}$, is extracted, then the component $i$ is assumed to be missing, and the tuple $(t_{i-1}, \texttt{null})$ is extracted.

The complexity of algorithm 1 depends on the complexity of computing $rep_i$ for candidate partial tuples at each step $i$. The computation of tuple representations is efficient. Indeed, leaf and node attributes are computed while reading the input document. Couple nodes attributes must be computed at each step of the incremental extraction process. They are computed only from extracted tuples at the previous step. Calculating node couples attributes involves finding the nearest common ancestor of a pair of nodes. This can be done in constant time after a linear preprocessing of the input tree structured document [8]. It should also be noted that a classifier $c_i$ may only use a subset of attributes reducing the computation time for the extraction process.

**Algorithm 1** Extraction algorithm

**Input:** document $d$; $n$ classifiers $c_1, \ldots, c_n$ according to representation schemas $rep_1, \ldots, rep_n$

**Notation:** $L(d)$ is the set of leaves of $d$, $S_i$ is the set of candidate partial tuples of length $i$, $S_i^+$ is the set of selected partial tuples of length $i$.

1: $S_1 = \{(l) \mid l \in L(d)\}$;
2: $S_1^+ = \{t_1 \in S_1 \mid c_1(rep_1(t_1)) = +1\}$
3: **for** $i = 2$ **to** $n$ **do**
4: $\quad S_i = \{t_i = (t_{i-1}, l) \mid t_{i-1} \in S_{i-1}^+,\ l \in L(d)\}$
5: $\quad S_i^+ = \{t_i \in S_i \mid c_i(rep_i(t_i)) = +1\}$
6: $\quad S_i^+ = S_i^+ \cup \{(t_{i-1}, \texttt{null}) \mid t_{i-1} \in S_{i-1}^+,\ \forall l \in L(d),\ c_i(rep_i((t_i, l))) = -1\}$
7: **end for**

**Output:** the set of extracted $n$-ary tuples $S_n^+$

## 5 The induction process

Let us consider a target $n$-ary relation. The learning process is incremental similarly to the extraction process. Given a set $D$ of completely annotated documents, algorithm 2 loops from 1 to the number $n$ of components. At each step $i$, it computes a classifier $c_i$ for tuples of length $i$. The number of positive examples is bounded by the number of tuples in $\cup_{d \in S} S^+(d)$, that is the number of tuples to be extracted in $D$. On the opposite, the number of negative examples can be exponential in $n$. So we only use a subset of negative examples to train each classifier $c_i$. The selection of negative examples is driven by the schema of the extraction process as well: at each step $i$ of the process, candidate tuples are built from positive examples of step $i-1$. Negative examples are computed by the function Neg taking as input a document $d$ in $D$ and a set of tuples to be extracted from $d$. It is defined by:

$$\mathsf{Neg}(d, S) = \{(x, l) \notin S | l \in L(d), \exists l' \in L(d) | (x, l') \in S\}$$

where $L(d)$ is the set of leaves of the document $d$. Then, examples are represented via $rep_i$ and the base supervised classification algorithm $W$ is applied. Algorithm 2 outputs a sequence of classifiers. This sequence will be the input of algorithm 1 for the extraction process.

## 6 Interactive wrapper induction system

Algorithm 2 suffers from one main flaw: it requires a set of documents where all tuples to extract are annotated. In this section, we propose to adapt this algorithm for learning from partially annotated examples in order to plug it in an interactive system. The aim is to reduce the number of user interactions to build $n$-ary wrappers.

In an interactive process, an end user first labels some examples to extract. Then we enter a loop where the system

**Algorithm 2** Induction algorithm

**Input:** a sample $D = \cup\{d\}$ of annotated documents

**Notation:** $S^+(d)$ is the set of tuples to be extracted in $d$; $S_i^+(d)$ is the projection of $S^+(d)$ over the first $i$ components

1: **for** $i = 1$ **to** $n$ **do**
2: $S_i^+ = \cup_{d \in D} S_i^+(d)$
3: $S_i^- = \cup_{d \in D} \mathsf{Neg}(d, S_i^+(d))$
4: $c_i = W(rep_i(S_i^+), rep_i(S_i^-))$
5: **end for**

**Output:** the sequence $(c_1, \ldots, c_n)$ of classifiers

proposes a new hypothesis wrapper and the user corrects this hypothesis until a good one is built. We have integrated the interactive process in the wrapper induction system following the incremental approach developed so far: for every $i$, wrappers for partial tuples are constructed in an interactive process.

We simulate an end user by an oracle $U$. The oracle $U$ is given access to a pool of documents. Given as input a hypothesis wrapper $w_i = (c_1, \ldots, c_i)$ where each $c_j$ is a classifier for partial tuples of length $j$, the oracle $U(w_i)$ returns true if $w_i$ is correct and returns *new annotations* otherwise. We suppose a working document $dc$ and that $U$ returns new annotations according to the following scenario. If $U(w_i)$ does not return true and if $w_i$ is not correct on $dc$, $U$ returns a false negative example (a tuple not extracted by $w_i$ while it should be extracted) and a false positive example (a tuple extracted by $w_i$ while it should not be extracted) from $dc$. We should note that a false positive or a false negative may not exist. If $U(w_i)$ does not return true and $w_i$ is correct on $dc$, then $U$ returns a new document with two partial tuples of length $i$ to be extracted.

The interactive wrapper induction system is the algorithm 3. It iterates over the length of partial tuples. At each step $i$, it learns from a set of completely annotated documents $D_i$ and a current working document $dc$, which is partially annotated. In the working document $dc$, $S_i^+(dc)$ (respectively $S_i^-(dc)$) denotes the set of partial tuples of length $i$ to be extracted (respectively tuples not to be extracted) given by the oracle $U$.

When $U$ returns a new document, it is supposed, according to the above scenario, that the current wrapper is correct over $dc$. Then, we add $dc$ to the set of completely annotated documents and the new working document is the document returned by $U$.

The sets $S_i^+(dc)$ and $S_i^-(dc)$ are updated according to the annotations given by $U$.

Positive examples are partial tuples of length $i$ to be extracted in the documents of $D_i$ and in the set $S_i^+(dc)$.

Negative examples for completely annotated documents in $D_i$ are defined according to function Neg as in Algorithm 2.

**Algorithm 3** interactive wrapper induction system

**Notation:** an oracle $U$ is an end user simulator; $d(U(c_1, \ldots, c_i))$ is the document returned by $U$; $d^+(U(c_1, \ldots, c_i))$ is the set of tuples to be extracted returned by $U$; $d^-(U(c_1, \ldots, c_i))$ is the set of tuples not to be extracted returned by $U$;

1: **for** $i = 1$ **to** $n$ **do**
2: $D_i = \emptyset$ {set of completely annotated documents}
3: $c_i =$ `null` {classifier}
4: $dc =$ `null` {current working document}
5: **while** `not` $U(c_1, \ldots, c_i)$ **do**
6: **if** $d(U(c_1, \ldots, c_i)) \neq dc$ **then**
7: {a new document to be processed}
8: $D_i = D_i \cup \{dc\}$ {$dc$ is completely annotated}
9: $dc = d(U(c_1, \ldots, c_i))$; $S_i^+(dc) = S_i^-(dc) = \emptyset$
10: **end if**
11: $S_i^+(dc) = S_i^+(dc) \cup d^+(U(c_1, \ldots, c_i))$
12: $S_i^-(dc) = S_i^-(dc) \cup d^-(U(c_1, \ldots, c_i))$
13: $S_i^+ = (\cup_{d \in D_i} S_i^+(d)) \cup S_i^+(dc)$
14: $S_i^- = (\cup_{d \in D_i} \mathsf{Neg}(d, S_i^+(d))) \cup \mathsf{Negp}(i, dc, S_i^+(dc), S_i^-(dc))$
15: $c_i = W(rep_i(S_i^+), rep_i(S_i^-))$
16: **end while**
17: **end for**

**Output:** the sequence $(c_1, \ldots, c_n)$ of classifiers

Negative examples for $dc$ should be carefully defined because $dc$ is partially annotated. For instance, let us consider the case $i = 1$, that is the induction of the wrapper for the seed. In the first interaction: there are only two positive examples available for learning. Thus we have to guess negative examples. The function Neg, used in the case of completely annotated documents, would give us all leaves in the current document which are not explicitly labeled as positive and the classifier learned would classify positive only leaves which are already known as positive!

To compute negative examples in the working document $dc$, we define the function Negp. It takes as input the index $i$, the working document $dc$, and the sets $S_i^+(dc)$ and $S_i^-(dc)$. When $i = 1$, it returns all leaves in $S_1^-(dc)$ together with all leaves whose path in the DOM tree is not the path of any leaf in $S_1^+(dc)$. When $i > 1$, two cases are to be distinguished: the first $i - 1$ components may be factorized or not other several tuples of length $i$. In the first case, several tuples of length $i$ are constructed over the same partial tuple of length $i - 1$. The function Neg would generate false negative examples. Thus we need to guess which leaves may be good candidates to complete an $(i - 1)$-ary tuple. Therefore, Negp returns all partial tuples of length $i$ in $S_i^-(dc)$ together with all partial tuples of length $i$ such that the partial tuple of length $i - 1$ is in $S_{i-1}^+$ and the path

of the $i^{th}$ component is not the path of any $i^{th}$ component in $S_i^+(dc)$. In the second case, Negp returns all partial tuples of length $i$ in $S_i^-(dc)$ together with $\mathsf{Neg}(dc, S^+(dc))$.

Then the set of negative examples is computed in Line 14.

A classifier $c_i$ is induced with the base supervised learning algorithm $W$ updating the current partial wrapper $w_i = (c_1, \ldots, c_i)$. And the process iterates over calls to $U$ and over $i$ when $U$ agrees with the current partial wrapper.

## 7 Experimental results

**Evaluation Protocol** An extracted tuple is correct if all its component values are correct. A component value is correct if it matches exactly the target component value. Performance of our wrapper induction system is evaluated using the average of precision ($P$), recall ($R$) and f-measure ($F$). They are computed with cross-validation techniques. In all experiments, the base learner for binary classifiers is Quinlan's C5.0 [15].

**Experiments on RISE datasets** Data sets in RISE correspond to the first two cases according to Section 2. Table 1 presents the experimental results where TD is the average number of tuples per document, and D is the number of documents used in the induction process. In this section and in the next section as well, at most two documents are given to the learner.

On BIGBOOK, OKRA and S20, Algorithm 2 obtain 100 of f-measure. For the problem S1 with the target relation (*company*, *price*, *product*), WIEN achieves 100 of f-measure while our system does not achieve 100 of f-measure because parts of textual values in leaves are needed for correct extraction. In the IAF dataset, with the target relation (*name*, *email*, *lastupdate*, *organization*, *altname*, *serviceprovider*), three of the six components of the target relation have missing values. Our system outperforms LIPX [17] (which has 46 of f-measure) and WIEN which fails because it can not deal with missing values. On $n$-ary extraction tasks, we can not compare with systems like STALKER because it uses $n$ unary wrappers and does not evaluate on tuples recombination. Further more it uses a post-processing, defined by the end-user, to construct tuples from component values. On unary extraction tasks, our system reaches the performances of STALKER (except on S1 as mentioned before). In particular on IAF, the f-measure of our system varies from 84 to 100, whereas it obtains 64 in the $n$-ary case. Indeed on IAF tuples recombination is more difficult than components extraction.

| Corpus | TD | D | $P$ | $R$ | $F$ |
|---|---|---|---|---|---|
| BIGBOOK | 18.29 | 1 | 100 | 100 | 100 |
| OKRA | 13,23 | 1 | 100 | 100 | 100 |
| S20 | 32.7 | 1 | 100 | 100 | 100 |
| S1 | 40.3 | 1 | 88.65 | 88.57 | 88.61 |
| IAF | 9 | 2 | 61.51 | 67.66 | 64.22 |

**Table 1. Experimental results on RISE datasets.**

| Corpus | Case | TD | D | $P$ | $R$ | $F$ |
|---|---|---|---|---|---|---|
| EXCITE 1 | 3,4 | 5 | 1 | 100 | 100 | 100 |
| BBC | 2,4 | 5 | 1 | 100 | 100 | 100 |
| BLS 1 | 3 | 29 | 1 | 100 | 100 | 100 |
| BLS 2 | 5 | 3.56 | 1 | 100 | 100 | 100 |
| BLS 3 | 3 | 6 | 1 | 100 | 100 | 100 |
| BLS 4 | 5 | 24 | 2 | 100 | 90 | 93.33 |
| BEA 1 | 4,3 | 8.30 | 2 | 97.11 | 88.58 | 92.58 |
| BEA 2 | 3 | 5.2 | 2 | 86.47 | 100 | 89.78 |
| BTS | 3 | 20 | 1 | 100 | 100 | 100 |

**Table 2. Experiments on corporate Web sites.**

**Experiments on Web datasets** We now consider datasets[6] obtained from several corporate Web sites. They were chosen to be representative of the tree organizations presented in Section 2. For the EXCITE 1 dataset, the relation to extract is (*town*, *day*, *weather*, *high*, *low*). There are five tuples, the component *town* is factorized while the others are stored in a rotated table. In BBC, the relation to extract is (*town*, *day*, *high*, *low*). The component *town* is factorized and the others are stored in a table. There are several datasets built from BLS with a target 3-ary relation. In BLS 1, BLS 2 and BLS 4, tuples are stored in a cross table, while in BLS 3 they are stored in a rotated table. In the BEA 1 benchmark (see Fig.1), the target relation is the 4-ary relation (*balance*,*year*,*country*,*export*). For each document of BEA 1, there are six tuples to extract. The component *country* is factorized among the six tuples. The other components are stored in a rotated table. In the BEA 2 benchmark, the target relation is the 3-ary relation (*year*,*balance*,*export*) stored in a rotated table. In the rotated table of BTS, the relation to extract is (*year*,*highway*,*oil*).

Experimental results are presented in Table 2. They show that Algorithm 2 succeeds with one or two completely annotated Web pages. Two pages were needed when there are large variations in the size of the tables and in the number of tuples to be extracted from one page to another.

**Experiments in the interactive system** We consider the interactive wrapper induction algorithm of Section 6 with its user simulator $U$. We compare the number of interactions in the completely annotated case and in the interactive framework. We denote by #F the number of interactions

[6] http://www.grappa.univ-lille3.fr/ marty/corpus.html

| Corpus | TD | # I | # P | # F |
|---|---|---|---|---|
| EXCITE 1 | 5 | 13 | 25 | 25 |
| BBC | 5 | 7 | 13 | 15 |
| BLS 1 | 29 | 25.8 | 32.40 | 87 |
| BLS 2 | 3.56 | 7.60 | 12.20 | 10.68 |
| BLS 3 | 6 | 13.60 | 21.60 | 18 |
| BLS 4 | 24 | 16.40 | 30.20 | 288 |
| BEA 1 | 8.30 | 32 | 56.40 | 66.40 |
| BEA 2 | 5.2 | 9.4 | 15.20 | 31.2 |
| BTS | 8.30 | 29 | 45.00 | 60 |

**Table 3. Interactive experiments**

performed by the user with algorithm 2. It is defined to be $n \times m$ where $m$ is the number of annotated tuples and $n$ is the arity of tuples. In the interactive system, we denote by #I the number of component values selected or unselected by the user $U$, and we denote by #P the number of interactions performed by the user $U$. The selection of an example requires one interaction for the seed and two interactions for a partial tuple of length $i$ (the tuple of length $i-1$ and the $i^{th}$ component value). Thus a correction requires: 1 or 2 interactions for the seed; 2 or 4 interactions for the next steps.

We present in table 3 the number of interactions required in the two scenarios to achieve perfect wrappers on corporate Web sites. The number #I of examples selected or unselected by the user in the interactive framework is smaller than the number #F of interactions for a complete annotation. The number of interactions #P in the interactive framework is smaller than #F on datasets with many tuples per document. For benchmarks with few tuples to extract per document, like BLS 2, #P can be greater than #F because negative examples must be selected and because many partial tuples must be selected per tuple to be extracted.

We made experiments where the components are taken in different orders and we do not notice significant variations of the performances.

## 8 Conclusion

We have presented a machine learning approach for $n$-ary relation extraction from semi-structured documents. Our algorithm combines the advantage that component extraction and tuples construction are made simultaneously in the wrapping procedure and in the induction process, without any post-processing.

The effectiveness of our approach is evident with completely annotated documents: we achieve state-of-the-art results on simple data organizations and also succeeds on intricate organizations like pivot table, rotated table and cross table where previous systems fail. Extraction and learning are fast. Furthermore, we have proposed an interactive framework in order to reduce the number of user actions and shown that our method is again improved: we are able to learn equally performant wrappers, with less effort for the user.

## References


[1] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ICDM*, p. 337–348, 2003.

[2] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *28th Int. VLDB Conference*, p. 119–128, 2001.

[3] J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. In *IJCAI Workshop on Grammatical Inference*, 2005.

[4] C. Chang and S. Lui. IEPAD: Information extraction based on pattern discovery. In *WWW*, 2001.

[5] W. Cohen, M. Hurst, and L. Jensen. *Web Document Analysis: Challenges and Opportunities*, chapter A Flexible Learning System for Wrapping Tables and Lists in HTML Documents. World Scientific, 2003.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, p. 109–118, 2001.

[7] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI*, p. 577–583, 2000.

[8] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[9] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521 – 538, 1998.

[10] L. S. Jensen and W. Cohen. Grouping extracted fields. In *ATEM Workshop, IJCAI*, 2001.

[11] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997.

[12] A. H. F. Laender, B. Ribeiro-Neto, A. S. Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[13] K. Lerman, C. A. Knoblock, and S. Minton. Automatic data extraction from lists and tables in web sources. In *ATEM Workshop, IJCAI*, 2001.

[14] I. Muslea, S. Minton, and C. Knoblock. Active learning with strong and weak views: a case study on wrapper induction. In *IJCAI*, p. 415–420, 2003.

[15] R. Quinlan. Data mining tools see5 and c5.0, 2004. http://www.rulequest.com/see5-info.html.

[16] S. Raeymaekers, M. Bruynooghe, and J. Van den Bussche. Learning (k,l)-contextual tree languages for information extraction. In *ECML*, v. 3720 of *LNAI*, p. 305–316, 2005.

[17] B. Thomas. Bottom-up learning of logic programs for information extraction from hypertext documents. In Springer-Verlag, editor, *In ECML/PKDD*, 2003.

[18] I. S. I. University of Southern California. Rise, a repository of online information sources used in information extraction tasks, 1998. http://www.isi.edu/info-agents/RISE/index.html.

[19] Y. Zhai and B. Liu. Extracting web data using instance-based learning. In *WISE*, p. 318–331, 2005.

[20] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, p. 76–85, 2005.

# Annexe F

# Learning Multi-label Alternating Decision Trees from Texts and Data

# Learning Multi-label Alternating Decision Trees from Texts and Data *

Francesco De Comité[1], Rémi Gilleron[2], and Marc Tommasi[2]
équipe Grappa — EA 3588

[1] Lille 1 University, 59655 Villeneuve d'Ascq Cedex, France
decomite@lifl.fr
[2] Lille 3 University, 59653 Villeneuve d'Ascq Cedex, France
gilleron,tommasi@univ-lille3.fr

**Abstract.** Multi-label decision procedures are the target of the supervised learning algorithm we propose in this paper. Multi-label decision procedures map examples to a finite set of labels. Our learning algorithm extends Schapire and Singer's Adaboost.MH and produces sets of rules that can be viewed as trees like Alternating Decision Trees (invented by Freund and Mason). Experiments show that we take advantage of both performance and readability using boosting techniques as well as tree representations of large set of rules. Moreover, a key feature of our algorithm is the ability to handle heterogenous input data: discrete and continuous values and text data.

**Keywords :** boosting - alternating decision trees - text mining - multi-label problems

## 1 Introduction

*When a patient spends more than 3 days in center X, measures of albuminuri as well as proteinuri are made. But if the patient is in center Y, then only measures of albuminuri are made.*

These sentences can be viewed as a multi-label classification procedure because more than one label among $\{albuminuri, proteinuri\}$ may be assigned to a given description of a situation. That is to say we are faced a categorization task for which the categories are not mutually exclusive. This work is originally motivated by a practical problem in medicine where each patient may be described by continuous-valued attributes (*e.g.* measures), nominal attributes (*e.g.* sex, smoker,...) and text data (*e.g.* descriptions, comments,...). It was important to produce rules that can be interpreted by physicians and also rules that reveal correlations between labels predictions. These requirements have lead to the realization of the algorithm presented in this paper[1].

* Partially supported by project DATADIAB: "ACI télémédecine et technologies pour la santé" and project TACT/TIC Feder & CPER Région-Nord Pas de Calais.

[1] The algorithm and a graphical user interface are available from http://www.grappa.univ-lille3.fr/grappa/index.php3?info=logiciels

Multi-label classification problems are ubiquitous in real world problems. Learning algorithms that can hold multi-label problems are therefore valuable. Of course there are many strategies to apply a combination of many binary classifiers to solve multi-label problems ([ASS00]). But most of them ignore correlations between the different labels. AdaBoost.MH algorithm proposed by Schapire and Singer ([SS00]) efficiently handles multi-label problems. For a given example, it also provides a real value as an outcome for each label. For practical applications, these values are important because they can be interpreted as a confidence rate about the decision for the considered label. AdaBoost.MH implements boosting techniques that are theoretically proved to transform a weak learner — called base classifier — into a strong one. The main idea of boosting is to combine many simple and moderately inaccurate rules built by the base classifier into a single highly accurate rule. The combination is a weighted majority vote over all simple rules. Boosting has been extensively studied and many authors have shown that it performs well on standard machine learning tasks ([Bre98], [FS96] [FS97]). Unfortunately, as pointed by Freund and Mason an others authors, the rule ultimately produced by a boosting algorithm may be difficult to understand and to interpret.

In [FM99], Freund and Mason introduce Alternating Decision Trees (ADTrees). The motivation of Freund and Mason was to obtain intelligible classification models when applying boosting methods. ADTrees are classification models inspired by both decision trees and option trees ([KK97]). ADTrees provide a symbolic representation of a classification procedure and give together with classification a measure of confidence. Freund and Mason also propose an alternating decision tree learning algorithm called ADTBoost in [FM99]. ADTBoost algorithm originates from two ideas: first, decision tree learning algorithms may be analyzed as boosting algorithms (see [DKM96], [KM96]); second, boosting algorithms could be used because ADTrees generalize both voted decision stumps and voted decision trees. Recently, the ADTree formalism has been extended to the multiclass case ([HPK$^+$02]).

We propose in this paper to extend ADTrees formalism to handle multi-label decision procedure. Decision procedures are learned from data described by nominal, continuous and text data. Our algorithm can be understood as an extension of AdaBoost.MH that permits a better readability of the classification rule ultimately produced as well as an extension to ADTBoost in order to handle multi-label classification problems. The multi-label ADTree formalism gives an intelligible set of rules viewable as a tree. Moreover, rules allow to combine atomic tests. In our implementation, we can handle test on (continuous or discrete) tabular data as well as tests on text data. This is particularly valuable in medicine where descriptions of patients combine diagnostic analysis, comments, dosages, measures, and so on. For instance, a rule can be built on both temperature and diagnostic, *if temperature* $>$ *37.5 and diagnostic contains "Cardiovascular" then ....* This kind of combination in a unique rule is not considered by others algorithms like Boostexter which implements AdaBoost.MH. We expect the algorithm to find more concise set of rules thanks to such combinations. We

are convinced that rules with several tests in their premises provide useful informations that can be interpreted by experts. In this paper, we only present results on freely available data sets. We compare our algorithm ADTBoost.MH with AdaBoost.MH on two data sets: the reuters collection and a new data set built from news articles[2].

In Section 2, we define multi-label problems and we recall AdaBoost.MH's functioning. Alternating Decision Trees are presented in Section 3. We define multi-label ADTrees which generalize ADTrees to the multi-label case in Section 4. An example is given in Figure 3. A Multi-label ADTree is an easily readable representation of both different ADTrees (one ADTree per label) and of many decision stumps (one per boosting round). We propose a multi-label ADTree learning algorithm ADTboost.MH based on both ADTboost and Adaboost.MH [SS98]. Experiments are given in Section 5. They show that our algorithm reaches the performance of well tuned algorithms like Boostexter.

## 2 Boosting and Multi-label Problems

Most of supervised learning algorithms deal with binary or multiclass classification tasks. In such a case, an instance belongs to one class and the goal of learning algorithms is to find an hypothesis which minimizes the probability that an instance is misclassified by the hypothesis. Even when a learning algorithm do not apply to the multiclass case, there exist several methods that can combine binary decision procedures in order to solve multiclass problems ([DB95], [ASS00]). In this paper, we consider the more general problem, called multi-label classification problem, in which an example may belong to any number of classes. Formally, let $\mathcal{X}$ be the universe and let us consider a set of labels $\mathcal{Y} = \{1, \ldots, k\}$. The goal is to find with input a sample $S = \{(x_i, Y_i) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}, 1 \leq i \leq m\}$ an hypothesis $h : \mathcal{X} \mapsto 2^{\mathcal{Y}}$ with low error. It is unclear to define the error in the multi-label case because different definitions are possible, depending on the application we are faced with. In this paper, we only consider the Hamming error.

*Hamming error:* The goal is to predict the set of labels associated with an example. Therefore, one takes into account *prediction errors* (an incorrect label is predicted) and *missing errors* (a label is not predicted). Let us consider a target function $c : \mathcal{X} \mapsto 2^{\mathcal{Y}}$ and an hypothesis $h : \mathcal{X} \mapsto 2^{\mathcal{Y}}$, the *Hamming error* of $h$ is defined by:

$$E_H(h) = \frac{1}{k} \sum_{l=1}^{k} D(\{x \in \mathcal{X} \mid (l \in h(x) \wedge l \notin c(x)) \vee (l \notin h(x) \wedge l \in c(x))\}). \quad (1)$$

[2] Data sets and perl scripts are available on `http://www.grappa.univ-lille3.fr/recherche/datasets`.

The factor $\frac{1}{k}$ normalizes the error in the interval $[0, 1]$. The training error over a sample $S$ is:

$$E_H(h, S) = \frac{1}{km} \sum_{i,l} (\| \ (l \in h(x_i) \wedge l \notin Y_i \ \| + \| \ l \notin h(x_i) \wedge l \in Y_i \ \|) \qquad (2)$$

where $\| \ a \ \|$ equals 1 if $a$ holds and 0 otherwise.

In the rest of the paper, we will consider learning algorithms that output mappings $h$ from $\mathcal{X} \times \mathcal{Y}$ into $\mathbb{R}$. The real value $h(x, l)$ can be viewed as a prediction value for the label $l$ for the instance $x$. Given $h$, we define a multi-label interpretation $h^m$ of $h$: $h^m$ is a mapping $\mathcal{X} \to 2^{\mathcal{Y}}$ such that $h^m(x) = \{l \in \mathcal{Y} \mid h(x, l) > 0\}$.

### AdaBoost.MH

Schapire and Singer introduce AdaBoost.MH in [SS00]. Originally, the algorithm supposes a weak learner from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$. In this section we focus on boosting decision stumps. We are given a set of conditions $\mathcal{C}$. Weak hypotheses are therefore rules of the form *if* $c$ *then* $(a_l)_{l \in \mathcal{Y}}$ *else* $(b_l)_{l \in \mathcal{Y}}$ where $c \in \mathcal{C}$ and $a_l, b_l$ are real values for each label $l$. Weak hypotheses therefore make their predictions based on a partitioning of the domain $\mathcal{X}$.

The AdaBoost.MH learning algorithm is given in Algorithm 1. Multi-label data are firstly transformed into binary data. Given a set of labels $Y \subseteq \mathcal{Y}$, let us define $Y[l]$ to be $+1$ if $l \in Y$ and to be $-1$ if $l \notin Y$. Given an input sample of $m$ examples, the main idea is to replace each training example $(x_i, Y_i)$ by $k$ examples $((x_i, l), Y_i[l])$ for $l \in \mathcal{Y}$. AdaBoost.MH maintains a distribution over $\mathcal{X} \times \mathcal{Y}$. It re-weights the sample at each boosting step. Basically, examples that were misclassified by the hypothesis in the previous round have an higher weight in the current round. It is proved in [SS98] that the normalization factor $Z_t$ induced by the re-weighting realizes a bound on the empirical Hamming loss of the current hypothesis. Therefore, this bound is used to guide the choice of the weak hypothesis at each step. AdaBoost.MH tries to minimize error while minimizing the normalization factor denoted by $Z_t$ at each step.

Let us denote $W_+^l(c)$ (resp. $W_-^l(c)$) the sum of weights of the positive (resp. negative) examples that satisfies condition $c$ and have label $l$.

$$W_+^l(c) = \sum_{i=1}^{i=m} D_t(i, l) \ \| \ x_i \text{ satisfies } c \wedge Y_i[l] = +1 \ \|$$

$$W_-^l(c) = \sum_{i=1}^{i=m} D_t(i, l) \ \| \ x_i \text{ satisfies } c \wedge Y_i[l] = -1 \ \|$$

Therefore, one can prove analytically that the the best prediction values $a_l$ and $b_l$ which minimize $Z_t$ at each step are:

$$a_l = \frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)} \ ; \ b_l = \frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)} \qquad (3)$$

leading to a normalization factor $Z_t$ :

$$Z_t(c) = 2 \sum_{l=1}^{l=k} \sqrt{W_+^l(c) W_-^l(c)} \quad (4)$$

**Algorithm 1** AdaBoost.MH($T$) where $T$ is the number of boosting rounds

**Input:** a sample $S = \{(x_1, Y_1), \ldots, (x_m, Y_m) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y} = \{1, \ldots, k\}\}$; $\mathcal{C}$ is a set of base conditions

1: Transform $S$ into $S^k = \{((x_i, l), Y_i[l]) \mid 1 \leq i \leq m, l \in \mathcal{Y}\} \subseteq (\mathcal{X} \times \mathcal{Y}) \times \{-1, +1\}$
2: Initialize the weights: $1 \leq i \leq m$, $l \in \mathcal{Y}$, $w_1(i, l) = 1$
3: **for** $t = 1..T$ **do**
4: choose $c$ which minimize $Z_t(c)$ according to Equation 4
5: build the rule $r_t$ : if $c$ then $\frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)}$ else $\frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)}$
6: Update weights : $w_{t+1}(i, l) = w_t(i, l) e^{-Y_i[l] r_t(x_i, l)}$
7: **end for**

**Output:** $f(x, l) = \sum_{t=1}^{T} r_t(x, l)$.

## 3 Alternating Decision Trees

Alternating Decision Trees (ADTrees) are introduced by Freund and Mason in [FM99]. They are similar to option trees developed by Kohavi *et al* [KK97]. An important motivation of Freund and Mason was to obtain intelligible classification models when applying boosting methods. Alternating decision trees contain *splitter nodes* and *prediction nodes*. A splitter node is associated with a test, a prediction node is associated with a real value. An example of ADTree is given in Figure 1. It is composed of four splitter nodes and nine prediction nodes. An instance defines a set of paths in an ADTree. The classification which is associated with an instance is the sign of the sum of the predictions along the paths in the set defined by this instance. Consider the ADTree in Figure 1 and the instance $x = (color = red, year = 1989, \ldots)$, the sum of predictions is $+0.2 + 0.2 + 0.6 + 0.4 + 0.6 = +2$, thus the classification is $+1$ with high confidence. For the instance $x = (color = red, year = 1999, \ldots)$, the sum of predictions is $+0.4$ and the classification is $+1$ with low confidence. For the instance $x = (color = white, year = 1999, \ldots)$, the sum of predictions is $-0.7$ and the classification is $-1$ with medium confidence.

ADTree depicted in Fig. 1 can also be viewed as consisting of a root prediction node and four units of three nodes each. Each unit is a decision rule and is composed of a splitter node and two prediction nodes that are its children. It is easy to give another description of ADTrees using sets of rules. For instance, the ADTree in Figure 1 is described by the set of rules:

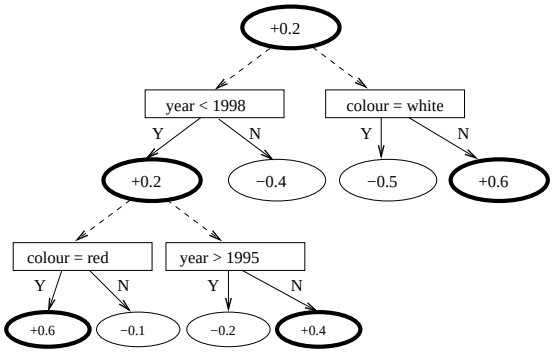



**Fig. 1.** an example of ADTree; bold prediction nodes define the set of nodes associated with the instance $x = (color = red, year = 1989, \ldots)$

| | | | | |
|---|---|---|---|---|
| If TRUE | then (if TRUE | then +0.2 | else 0 | ) else 0 |
| If TRUE | then (if year $<$ 1998 | then +0.2 | else -0.4 | ) else 0 |
| If year $<$ 1998 | then (if colour $=$ red | then +0.6 | else -0.1 | ) else 0 |
| If year $<$ 1998 | then (if year $>$ 1995 | then -0.2 | else +0.4 | ) else 0 |
| If TRUE | then (if colour $=$ white | then -0.5 | else +0.6 | ) else 0 |

### ADTBoost

Rules in an ADTree are similar to decision stumps. Consequently, one can apply boosting methods in order to design an ADTree learning algorithm. The algorithm proposed by Freund and Mason [FM99] is based on this idea. It relies on Schapire and Singer [SS98] study of boosting methods. A rule in an ADTree defines a partition of the instance space into three blocks defined by $C_1 \wedge c_2$, $C_1 \wedge \neg c_2$ and $\neg C_1$. Following this observation, Freund and Mason apply a variant of Adaboost proposed in [SS98] for domain-partitioning weak hypotheses. Basically, the learning algorithm builds an ADTree with a top down strategy. At every boosting step, it selects and adds a new rule or equivalently a new unit consisting of a splitter node and two prediction nodes. Contrary to the case of decision trees, the reader should note that a new rule can be added below any prediction node in the current ADTree.

Freund and Mason's algorithm ADTboost is given as Algorithm 2. Similarly to Adaboost algorithm, ADTboost updates weights at each step (line 8). The quantity $Z_t(C_1, c_2)$ is a normalization coefficient. It is defined by

$$Z_t(C_1, c_2) = 2\left(\sqrt{W_+(C_1 \wedge c_2)W_-(C_1 \wedge c_2)} + \sqrt{W_+(C_1 \wedge \neg c_2)W_+(C_1 \wedge \neg c_2)}\right) + W(\neg C_1) \quad (5)$$

where $W_+(C)$ (resp. $W_-(C)$) is the sum of the weights of the positive (resp. negative) examples that satisfy condition $C$. It has been shown that the product of all such coefficients gives an upper bound of the training error. Therefore, ADTboost selects a precondition $C_1$ and a condition $c_2$ that minimize $Z_t(C_1, c_2)$ (line 5) in order to minimize the training error. The prediction values $a$ and $b$ (line 6) are chosen according to results in [SS98].

ADTboost is competitive with boosting decision tree learning algorithms such as C5 + Boost. Moreover, the size of the ADTree generated by ADTboost is often smaller than the model generated by other methods. These two points have strongly motivated our choices although ADTboost suffers from some drawbacks *e.g.* the choice of the number of boosting rounds is difficult and overfitting occurs. We discuss these problems in the conclusion.

**Algorithm 2** ADTboost($T$) where $T$ is the number of boosting rounds

**Input:** a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m) \mid x_i \in \mathcal{X}, y_i \in \{-1, +1\}\}$; a set of base conditions $\mathcal{C}$.

1: Initialize the weights: $1 \leq i \leq m$, $w_{i,1} = 1$
2: Initialize the ADTree: $\mathcal{R}_1 = \{r_1 : (\text{if } \boldsymbol{T} \text{ then } (\text{if } \boldsymbol{T} \text{ then } \frac{1}{2} \ln \frac{W_+(\boldsymbol{T})}{W_-(\boldsymbol{T})}) \text{ else } 0) \text{ else } 0)\}$
3: Initialize the set of preconditions: $\mathcal{P}_1 = \{\boldsymbol{T}\}$
4: **for** t = 1..$T$ **do**
5: Choose $C_1 \in \mathcal{P}_t$ and $c_2 \in \mathcal{C}$ which minimize $Z_t(C_1, c_2)$ according to Equation 5
6: $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_{t+1}$: (if $C_1$ then (if $c_2$ then $\frac{1}{2} \ln \frac{W_+(C_1 \wedge c_2)}{W_-(C_1 \wedge c_2)}$) else $\frac{1}{2} \ln \frac{W_+(C_1 \wedge \neg c_2)}{W_-(C_1 \wedge \neg c_2)}$) else 0)$\}$
7: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{C_1 \wedge c_2, C_1 \wedge \neg c_2\}$
8: update weights: $w_{i,t+1}(i) = w_{i,t}(i) e^{-y_i r_t(x_i)}$
9: **end for**

**Output:** ADTree $\mathcal{R}_{T+1}$

## 4 Multi-label Alternating Decision Trees

### Multi-label ADTrees

We generalize ADTrees to the case of multi-label problems. A prediction node is now associated with a set of real values, one for each label. An example of such an ADTree is given in Figure 3.

Let $\mathcal{X}$ be the universe, the conjunction is denoted by $\wedge$, the negation is denoted by $\neg$, and let $\boldsymbol{T}$ be the `True` condition. Let $(0)_{l \in \mathcal{Y}}$ be a vector of $l$ zeros.

**Definition 1.** *Let $\mathcal{C}$ be a set of base conditions where a base condition is a boolean predicate over instances. A precondition is a conjunction of base conditions and negations of base conditions. A rule in an ADTree is defined by a precondition $C_1$, a condition $c_2$ and two vectors of real numbers $(a_l)_{l \in \mathcal{Y}}$ and $(b_l)_{l \in \mathcal{Y}}$:*

*if* $C_1$ *then (if* $c_2$ *then* $(a_l)_{l\in\mathcal{Y}}$ *else* $(b_l)_{l\in\mathcal{Y}}$*) else* $(0)_{l\in\mathcal{Y}}$*,*

*A multi-label alternating decision tree (multi-label ADTree) is a set* $\mathcal{R}$ *of such rules satisfying properties (i) and (ii):*

*(i) the set* $\mathcal{R}$ *must include an initial rule for which the precondition* $C_1$ *is* $\boldsymbol{T}$*, the condition* $c_2$ *is* $\boldsymbol{T}$*, and* $b$ *equals* $(0)_{l\in\mathcal{Y}}$*;*

*(ii) whenever the set* $\mathcal{R}$ *contains a rule with a precondition* $C_1'$*,* $\mathcal{R}$ *also contains another rule with precondition* $C_1$ *and there is a base condition* $c_2$ *such that either* $C_1' = C_1 \wedge c_2$ *or* $C_1' = C_1 \wedge \neg c_2$*.*

A multi-label ADTree maps each instance to a vector of real number in the following way:

**Definition 2.** *A rule r: if* $C_1$ *then (if* $c_2$ *then* $(a_l)_{l\in\mathcal{Y}}$ *else* $(b_l)_{l\in\mathcal{Y}}$ *) else* $(0)_{l\in\mathcal{Y}}$ *associates a real value* $r(x,l)$ *with any* $(x,l) \in \mathcal{X}\times\mathcal{Y}$*. If* $(x,l)$ *satisfies* $C = C_1 \wedge c_2$ *then* $r(x,l)$ *equals* $a_l$*; if* $(x,l)$ *satisfies* $C = C_1 \wedge \neg c_2$ *then* $r(x,l)$ *equals* $b_l$*; otherwise,* $r(x,l)$ *equals* $0$*.*

*An ADTree* $\mathcal{R} = \{r_i\}_{i\in I}$ *associates a prediction value* $R(x,l) = \sum_{i\in I} r_i(x,l)$ *with any* $(x,l) \in \mathcal{X}\times\mathcal{Y}$*. A multi-label classification hypothesis is associated with* $H$ *defined by* $H(x,l) = sign(R(x,l))$ *and the real number* $|R(x,l)|$ *is interpreted as the confidence assigned to* $H(x,l)$*,* i.e. *the confidence assigned to the label* $l$ *for the instance* $x$*.*

### ADTBoost.MH

Our multi-label alternating decision tree learning algorithm is derived from both ADTboost and AdaBoost.MH algorithms. Following [SS98], we now make precise the calculation of the prediction values and the value of the normalization factor $Z_t(C_1, c_2)$. The reader should note that we consider partitions over $\mathcal{X}\times\mathcal{Y}$.

On round $t$, let us denote the current distribution over $\mathcal{X}\times\mathcal{Y}$ by $D_t$, and let us consider $W_+^l(C)$ (resp. $W_-^l(C)$) as the sum of the weights of the positive (resp. negative) examples that satisfy condition $C$ and have label $l$:

$$W_+^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \wedge Y_i[l] = +1 \parallel$$

$$W_-^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \wedge Y_i[l] = -1 \parallel$$

$$W^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \parallel$$

It is easy to prove that the normalization factor $Z_t$ and the best prediction values $a_l$ and $b_l$ are:

$$a_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge c_2)}{W_-^l(C_1 \wedge c_2)} \; ; \; b_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge \neg c_2)}{W_-^l(C_1 \wedge \neg c_2)} \tag{6}$$

$$Z_t(C_1, c_2) = 2 \sum_{l=1}^{l=k} \left[ \sqrt{W_+^l(C_1 \wedge c_2) W_-^l(C_1 \wedge c_2)} + \sqrt{W_+^l(C_1 \wedge \neg c_2) W_-^l(C_1 \wedge \neg c_2)} \right] + W(\neg C_1) \tag{7}$$

The algorithm ADTboost.MH is given as Algorithm 3. In order to avoid extrem values for confidence values, we use the following formulas:

$$a_l = \frac{1}{2} ln \frac{W_{+1}^l(C_1 \wedge c_2) + \epsilon}{W_{-1}^l(C_1 \wedge c_2) + \epsilon} \; ; \; b_l = \frac{1}{2} ln \frac{W_{+1}^l(C_1 \wedge \neg c_2) + \epsilon}{W_{-1}^l(C_1 \wedge \neg c_2) + \epsilon} \tag{8}$$

with $\epsilon = \frac{1}{2mk}$.

**Algorithm 3** ADTboost.MH($T$) where $T$ is the number of boosting rounds

**Input:** a sample $S = \{(x_1, Y_1), \ldots, (x_m, Y_m) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y} = \{1, \ldots, k\}\}$; $\mathcal{C}$ is a set of base conditions

1: Transform $S$ into $S^k = \{((x_i, l), Y_i[l]) \mid 1 \leq i \leq m, l \in \mathcal{Y}\} \subseteq (\mathcal{X} \times \mathcal{Y}) \times \{-1, +1\}$
2: Initialize the weights: $1 \leq i \leq m$, $l \in \mathcal{Y}$, $w_1(i, l) = 1$
3: Initialize the multi-label ADTree:
$\mathcal{R}_1 = \{r_1 : (\text{if } \boldsymbol{T} \text{ then } (\text{if } \boldsymbol{T} \text{ then } (a_l = \frac{1}{2} \ln \frac{W_+^l(\boldsymbol{T})}{W_-^l(\boldsymbol{T})})_{l \in \mathcal{Y}}) \text{ else } (b_l = 0)_{l \in \mathcal{Y}}) \text{ else } 0)\}$.
4: Initialize the set of preconditions: $\mathcal{P}_1 = \{\boldsymbol{T}\}$.
5: **for** $t = 1..T$ **do**
6: choose $C_1 \in \mathcal{P}_t$ and $c_2 \in \mathcal{C}$ which minimize $Z_t(C_1, c_2)$ according to Equation 7
7: $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_{t+1} : (\text{if } C_1 \text{ then } (\text{if } c_2 \text{ then } (a_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge c_2)}{W_-^l(C_1 \wedge c_2)})_{l \in \mathcal{Y}}) \text{ else } (b_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge \neg c_2)}{W_-^l(C_1 \wedge \neg c_2)})_{l \in \mathcal{Y}}) \text{ else } 0)\}$
8: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{C_1 \wedge c_2, C_1 \wedge \neg c_2\}$
9: Update weights : $w_{t+1}(i, l) = w_t(i, l) e^{-Y_i[l] r_t(x_i, l)}$
10: **end for**

**Output:** multi-label ADTree $\mathcal{R}_{T+1}$

### Relations with other formalisms

Multi-label ADTrees trivially extends several common formalisms in machine learning.

*Voted decision stumps* produced for instance by AdaBoost.MH presented as Algorithm 1 are obviously multi-label ADTrees where each rule has $\boldsymbol{T}$ as a precondition. These "flat" multi-label ADTrees can be the output of ADTBoost.MH

if we impose a very simple control. Line 6 of Algorithm 3, we choose $c_2$ in $\mathcal{C}$ which minimize $Z_t(\boldsymbol{T}, c_2)$ according to Equation 7. Thus, AdaBoost.MH can be considered as a parameterization of ADTBoost.MH.

*(Multi-label) decision trees.* A (multi-label) decision tree is an ADTree with the following restrictions: any inner prediction node contains 0; there is at most one splitter node below every prediction node; prediction nodes at a leaf position contain values that can be interpreted as classes (using for instance the sign function in the binary case).

*Weighted vote of (Multi-label) decision trees.* Voted decision trees $t_1, \ldots, t_k$ associated with weights $w_1, \ldots, w_k$ are also simply transformed into ADTrees. One needs to add prediction nodes containing the weight $w_i$ at every leaf of the tree $t_i$ and graft all trees at the root of an ADTree.

## 5 Experiments

In this section, we describe the experiments we conduct with ADTBoost.MH. We mainly argue that we can obtain both accurate and readable classifiers over tabular and text data using multi-label alternating decision trees.

### Implementation

Our implementation of ADTBoost.MH supports items descriptions that include discrete attributes, continuous attributes and texts. In the case of a discrete attribute $A$ whose domain is $\{v_1, \ldots, v_n\}$, the set of base conditions are binary conditions of the form $A = v_i$, $A \neq v_i$. In the case of a continuous attribute $A$, we consider binary conditions of the form $A < v_i$. Finally, in the case of text-valued attributes over a vocabulary $V$, base conditions are of the form *$m$ occurs in $A$* where $m$ belongs to $V$.

Missing values are handled in ADTBoost.MH as in Quinlan's C4.5 software ([Qui93]). Let us consider an ADTree $R$, a position $p$ in $R$ associated with condition $c$ based on an attribute $A$ and an instance for which the value of $A$ is missing. We estimate probabilities for the assertions $c$ to be true or false, based on the observation of the training set. The obtained values are assigned to the missing value of this instance and then propagated below $p$.

### Data sets

*The Reuters collection* is the most commonly-used collection for text classification. We use a formatted version of Reuters version 2 (also called Reuters-21450) prepared by Y. Yang and colleagues[3]. Documents are labeled to belong to at least one of the 135 possible categories. A "sub-category" relation governs categories.

[3] available at `http://moscow.mt.cs.cmu.edu:8081/reuters_21450/parc/`

Nine of them constitute the top level of this hierarchy. Because most of the articles in the whole Reuters data set belong to exactly one category, in the experiments we select categories and articles for which overlaps between categories are more significant.

*News collection* We prepare a new data set from newsgroups archives in order to build a multilabel classification problem where cases are described by texts, continuous and nominal values. We obtain from `ftp://ftp.cs.cmu.edu/user/ai/pubs/news/comp.ai/` news articles posted in the comp.ai newsgroup in july 1997. Some articles in this forum have been cross posted in several newsgroups. The classification task consists in finding in which newsgroup a news has been cross posted.

Each news is described by seven attributes. Two of them are textual data: the subject and the text of the news. Four attributes are continuous (natural numbers): the number of lines in the text of the news, the number of references (that is the number of parents in the thread discussion), the number of capitalized words and the number of words in the text of the news. One attribute is discrete: the top level domain of sender's email address. We have droped small words, less than three letters, and non purely alphabetic words (*e.g.* R2D2)[4]. There are 524 articles and we keep only the five most frequent cross posted newsgroups as labels. The five newsgroups are: misc.writing (61 posts), sci.space.shuttle (68 posts), sci.cognitive (70 posts), rec.arts.sf.written (70 posts) and comp.ai.philosophy (73 posts). Only 171 articles were cross posted to at least one of these five newsgroups (60 in one, 51 in two, 60 in three).

### Results

We first train our algorithm on the Reuters dataset in order to evaluate it against Boostexter (Available implementation of AdaBoost.MH [5]). Reuters dataset consists in a train set and a test set. But following the protocol explained in [SS00], we merge these two sets and we focused on the nine topics constituting the top hierarchy. We then select the subsets of the $k$ classes with the largest number of articles for $k = 3 \ldots 9$. Results were computed on a 3-fold cross-validation. The number of boosting steps being set to 30. We report in table 1 one-error, coverage and average precision for Boostexter and ADTBoost.MH.

In the news classification problem, ranks of labels are less relevant. We only report the hamming error. Our algorithm ADTBoost.MH builds rules that may have large preconditions. This feature allows to partition the space in a very fine way and the training error can decrease very quickly. This can be observed on the news data set. After 30 boosting steps, the training error is 0 for the model generated by ADTBoost.MH. This is achieved by Boostexter after 230 boosting steps. On the one hand, smaller models can be generated by ADTBoost.MH. On

[4] Data sets and perl scripts are available on `http://www.grappa.univ-lille3.fr/recherche/datasets`.

[5] `http://www.cs.princeton.edu/~schapire/boostexter.html`

| | ADTree | | | Boostexter | | |
|---|---|---|---|---|---|---|
| $k$ | Error | Cover | Prec | Error | Cover | Prec |
| 3 | 6.01% | 0.07 | 0.97 | 6.48% | 0.08 | 0.97 |
| 4 | 7.03% | 0.10 | 0.96 | 7.93% | 0.11 | 0.96 |
| 5 | 8.31% | 0.12 | 0.95 | 8.99% | 0.14 | 0.95 |
| 6 | 12.70% | 0.24 | 0.92 | 12.34% | 0.24 | 0.92 |
| 7 | 14.72% | 0.31 | 0.91 | 14.32% | 0.30 | 0.91 |
| 8 | 16.01% | 0.34 | 0.91 | 15.90% | 0.35 | 0.90 |
| 9 | 16.77% | 0.40 | 0.89 | 16.60% | 0.39 | 0.89 |

**Table 1.** Comparing Boostexter and ADTree on the Reuters data set. The number $k$ is the number of labels in the multi-label classification problem.

the other hand, both ADTBoost.MH and ADTBoost tend to overspecialize and this phenomenon seems to occur more quickly for ADTBoost.MH.

Table 2 reports the hamming error on the cross posted news data set computed with a ten-fold cross validation. Note that the hamming error of the procedure that associate the empty set of label to each case is 0.306.

| Boosting steps | ADTBoost.MH | Boostexter |
|---|---|---|
| 10 | 0.024 | 0.022 |
| 30 | 0.023 | 0.019 |
| 50 | 0.017 | 0.017 |
| 100 | 0.017 | 0.014 |

**Table 2.** Hamming error of ADTBoost.MH and Boostexter on the cross posted news data set.

Figure 2 shows an example of rules produced by ADTBoost.MH on this data set and its graphical representation is depicted in Fig. 3. Both representations allow to interpret the model generated by ADTBoost.MH. For instance, according to the weights computed in the five first rules, one may say that when an article is cross posted in sci.space.shuttle it is not in sci.cognitive. On the contrary rec.arts.sf.written seems to be correlated with sci.space.shuttle. Below is an example of a rule with conditions that mix tests over textual data and tests over continuous data.

```
If subject  (not contains  Birthday) and  (subject  not contains  Clarke)
   and (subject  not contains  Secrets)
                     Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
  If #lines >= 22.50 then       -0.37   -3.24   0.24    0.17    0.08
  If #lines < 22.50 then        -0.12   -2.88   -3.51   -0.41   -5.07
Else 0
```

```
Rule 0
if TRUE
 Class :                misc_w  sci_sp  sci_co  comp_a  rec_ar
   If TRUE then         -1.01   -0.95   -0.93   -0.91   -0.93
Else 0
-------------------------
Rule 1
if TRUE
                           Class :      misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Birthday then       1.71    1.50    -6.35   -6.35   1.71
   If subject  not contains  Birthday then  -7.35   -2.02   -0.86   -0.84   -1.96
Else 0
-------------------------
Rule 2
If subject  not contains  Birthday
                           Class :      misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Emotional then      -2.93   -5.59   7.03    2.65    -5.62
   If subject  not contains  Emotional then -4.12   0.05    -0.41   -0.37   0.05
Else 0
-------------------------
Rule 3
If subject  not contains  Birthday
                              Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Clarke then         -0.46   6.92    -5.30   -5.33   6.89
   If subject  not contains  Clarke then    -2.31   -6.92   0.01    0.01    -1.10
Else 0
-------------------------
Rule 4
If subject  not contains  Birthday
If subject  not contains  Clarke
                              Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Secrets then        -0.17   -1.99   2.61    -5.83   -4.92
   If subject  not contains  Secrets then   -1.32   -3.59   -0.33   0.02    0.02
Else 0
```

**Fig. 2.** Output of ADTBoost.MH on the news data set

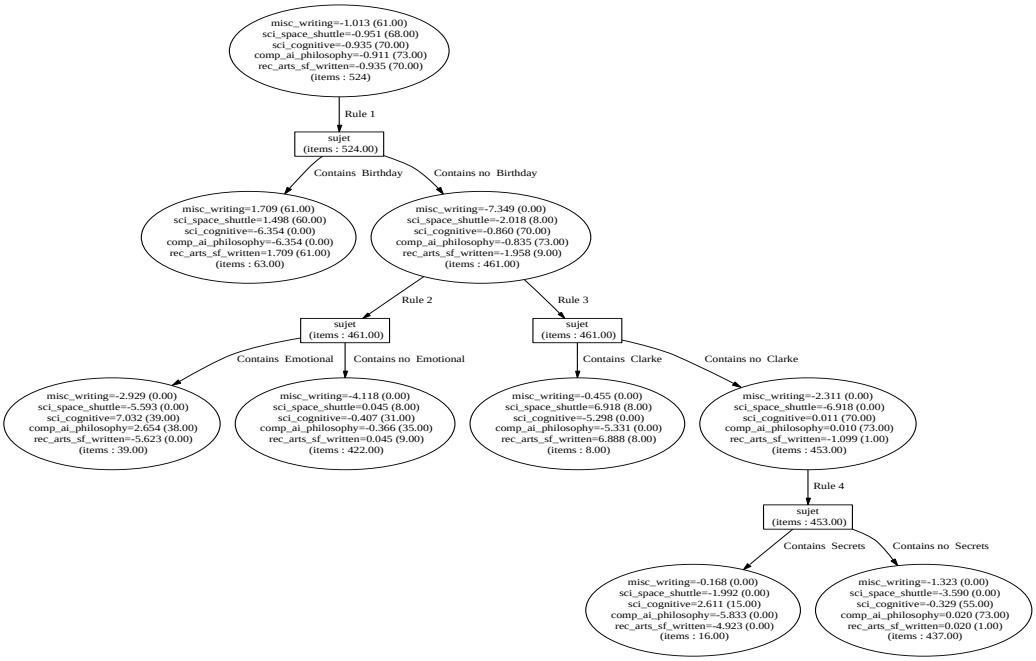



**Fig. 3.** A five rule multilabel ADTree built on the cross posted news data set.

## 6 Conclusion

We have proposed a learning algorithm ADTBoost.MH that can handle multi-label problems and produce intelligible models. It is based on boosting methods and seems to reach the performance of well tuned algorithms like AdaBoost.MH. Further works concern a closer analysis of the overspecialization phenomenon.

The number of rules in a multi-label ADTree is related to the number of boosting rounds in boosting algorithms like AdaBoost.MH. Readability of multi-label ADTree is clearly altered when the number of rules becomes large. But, this possibly large set of rules depicted as a tree comes with weights and with an ordering that permits "stratified" interpretations. Indeed, due to the algorithmic bias in the algorithm, rules that are firstly generated contribute to reduce the most the training error. Nonetheless, navigation tools and high quality user interfaces should be built to improve readability.

## References

[ASS00] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary:a unifying approach for margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 9–16, 2000.

[Bre98] L. Breiman. Combining predictors. Technical report, Statistic Department, 1998.

[DB95] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[DKM96] T. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proc. 13th International Conference on Machine Learning*, pages 96–104. Morgan Kaufmann, 1996.

[FM99] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133, 1999.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[HPK+02] G. Holmes, B. Pfahringer, R. Kirkby, E. Frank, and M. Hall. Multiclass alternating decision trees. In *Proceedings of the European Conference on Machine Learning*. Springer Verlag, 2002.

[KK97] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Proc. 14th International Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.

[KM96] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 459–468, 1996.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[SS98] Robert F. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*, pages 80–91, New York, July 24–26 1998. ACM Press.

[SS00] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

# Annexe G

# Text Classification from Positive and Unlabeled Examples

# Text Classification from Positive and Unlabeled Examples

**François Denis**
Équipe BDA, LIF
Marseille, FRANCE
fdenis@cmi.univ-mrs.fr

**Rémi Gilleron**
Équipe Grappa
Lille, FRANCE
gilleron@univ-lille3.fr

**Marc Tommasi**
Équipe Grappa
Lille, FRANCE
tommasi@univ-lille3.fr

**Abstract**

This paper shows that binary text classification is feasible with positive examples and unlabeled examples. This is important because in many text classification problems hand-labeling examples is expensive while examples of one class and unlabeled examples are readily available. We introduce a naive Bayes algorithm for learning from positive and unlabeled documents. Experimental results show that performance of our algorithm is comparable with naive Bayes algorithm for learning from labeled data.

**Keywords:** text mining, text classification, semi-supervised learning, positive data.

## 1 Introduction

Recently there has been significant interest in text learning algorithms that combine information from labeled and unlabeled data. For the labeled data, supervised learning algorithms apply, but their performance can be poor for a small labeled data set and they cannot take advantage of the unlabeled data. For the unlabeled data, unsupervised learning algorithms apply, but they do not use the labels. Thus, learning with labeled and unlabeled data – sometimes named as semi-supervised learning – falls between supervised and unsupervised learning. This research area is motivated by the fact that it is often tedious and expensive to hand-label large amount of training data, specially for text learning tasks, while unlabeled data are freely available.

Several learning algorithms have been defined for text learning tasks in the semi-supervised setting. We only consider supervised learning algorithms with the help of unlabeled data. Such approaches include using Expectation Maximization to estimate maximum a posteriori parameters [11], using transductive inference for support vector machines [5], using the unlabeled data to define a metric or a kernel function [4], using a partition of the set of features into two disjoint sets [1, 10].

We address the issue of learning from positive and unlabeled data where positive data are examples of one fixed target class. We have given in previous papers theoretical and experimental results [2, 7]: we have proven that every class learnable in the Statistical Query model [6] is learnable from positive statistical queries (estimates of probabilities over positive instances) and instance statistical queries (estimates of probabilities over the instance space) when a lower bound on the positive class probability is given; we have also designed a decision tree induction algorithm from positive and unlabeled examples.

In the present paper, we design text learning algorithms from positive and unlabeled documents. Let us consider two examples of applications. A first example is learning to classify web pages as “interesting” for a specific user. His bookmarks define a set of positive examples because they correspond to interest-

ing web pages for this user. Unlabeled examples are easily available on the World Wide Web. A second example is mail filtering. For a given mailing list and a specific user, positive examples are mails from the mailing list which have been saved by the user in his mailboxes. Again, unlabeled examples can easily be obtained by storing all mails from the mailing list, say during one week. It is interesting to note that no hand-labeling is needed in our framework.

In Section 2, we design a naive Bayes algorithm from positive and unlabeled examples. The key step is in estimating word probabilities for the negative class because negative examples are not available. This is possible according to the following assumption: an estimate of the positive class probability (the ratio of positive documents in the set of all documents) is given as input to the learner. In practical situations, the positive class probability can be empirically estimated or provided by domain knowledge.

In Section 3, we give experimental results on the WebKB Course data set [1]. The results show that error rates of naive Bayes classifiers obtained from $p$ positive examples completed with enough unlabeled examples are lower than error rates of naive Bayes classifiers obtained from $p$ labeled documents. The experiments suggest that positive examples may have a high value in context of semi-supervised learning.

## 2 Naive Bayes from positive and unlabeled examples

### 2.1 Naive Bayes

Naive Bayes classifiers are commonly-used in text classification [8]. The basic idea is to use the joint probabilities of words and classes to estimate the probabilities of classes given a document. The naive part is the assumption that the presence of each word in a document is conditionnally independent of all other words in the document given its class. This conditional independence assumption is clearly violated in real-world problems. Nevertheless, Naive Bayes classifiers are among the most effective text classification systems [3, 9].

We only consider binary classification problems with a set of classes $\{0,1\}$ where 1 corresponds to the *positive class*. We consider *bag-of-words* representations for documents. Naive Bayes is given in Table 1. It assumes an underlying generative model. In this model, first a class is selected according to class prior probabilities. A document length is chosen independently of the class. Then, the generator creates each word in a document by drawing from a multinomial distribution over words specific to the class.

Given a vocabulary $V$ and a set $D$ of labeled documents, let us denote by $PD$ (respectively $ND$) the set of positive documents (respectively negative documents) in the set $D$. The class probabilities $P(c)$ are estimated by:

$$\hat{P}(0) = \frac{Card(ND)}{Card(D)}; \hat{P}(1) = \frac{Card(PD)}{Card(D)} \quad (1)$$

where $Card(X)$ is the cardinality of set $X$.

A key step in implementing naive Bayes is estimating the word probabilities $Pr(w_i|c)$. The word probabilities $Pr(w_i|c)$ are estimated by counting the frequency that word $w_i$ occurs in all word occurrences for documents in class $c$:

$$\hat{Pr}(w_i|0) = \frac{N(w_i, ND)}{N(ND)}$$

$$\hat{Pr}(w_i|1) = \frac{N(w_i, PD)}{N(PD)}$$

where $N(w_i, X)$ is the total number of times word $w_i$ occurs in the documents in the set $X$ and $N(X)$ the total number of word occurrences in set $X$. A document cannot be classified as a member of class $c$ as soon as it contains a word $w$ which does not occur in any labeled document of class $c$. To make the probability estimates more robust with respect to infrequently encountered words, smoothing methods are used or equivalently a prior distribution over multinomials is assumed. We

consider the classical *Laplace smoothing*, and the class probability estimates are:

$$\hat{Pr}(w_i|0) = \frac{1 + N(w_i, ND)}{Card(V) + N(ND)} \quad (2)$$

$$\hat{Pr}(w_i|1) = \frac{1 + N(w_i, PD)}{Card(V) + N(PD)} \quad (3)$$

We now give formulas which are needed in the next section. We can write the following equation:

$$Pr(w_i) = Pr(w_i|0)Pr(0) + Pr(w_i|1)Pr(1) \quad (4)$$

where $Pr(w_i)$ is the probability that the generator creates $w_i$ and $Pr(1)$ is the probability that the generator creates a word in a positive document. Let us suppose that we are given a set $D = PD \cup ND$ of labeled documents. An estimate of $Pr(w_i)$ is $\frac{N(w_i,D)}{N(D)}$. An estimate of $Pr(1)$ is $\frac{N(PD)}{N(D)}$. But, under the assumption that the lengths of documents are independent of the class, another estimate of $Pr(1)$ is $\hat{P}(1) = \frac{Card(PD)}{Card(D)}$.

Table 1: Naive Bayes from labeled documents (NB)

Given a set $D$ of labeled documents, the naive Bayes classifier classifies a document $d$ consisting of $n$ words $(w_1, \ldots, w_n)$ – with possibly multiple occurrences of a word $w$ – as a member of the class

$$\mathsf{NB}(d) = \underset{c \in \{0,1\}}{\operatorname{argmax}} \hat{P}(c) \prod_{i=1}^{i=n} \hat{Pr}(w_i|c) \quad (5)$$

where the class probability estimates are calculated according to Equations 1 and the word probability estimates are calculated according to Equations 2 and 3.

### 2.2 Naive Bayes from positive and unlabeled examples

In the present section, training data consist of a set $PD$ of positive documents together with a set $UD$ of unlabeled documents. The key point is to compute sufficiently accurate probability estimates in Equation 5 from positive and unlabeled data only. We assume that an estimate $\hat{P}(1)$ of the positive class probability $P(1)$ is given to the learner. Then, an estimate of the negative class probability is setting $\hat{P}(0)$ to $1 - \hat{P}(1)$. The key step is estimating the word probabilities.

#### Estimating Word Probabilities

Let us consider that we are given an estimate $\hat{P}(1)$ of the positive class probability $P(1)$, a set $PD$ of positive documents together with a set $UD$ of unlabeled documents.

The positive word probability estimates are calculated using Equation 3 with the input set $PD$ of positive documents.

For the negative word probabilities, from Equation 4, we derive the following equation:

$$Pr(w_i|0) = \frac{Pr(w_i) - Pr(w_i|1) \times Pr(1)}{1 - Pr(1)} \quad (6)$$

We use this equation in order to derive the negative word probability estimates. In Equation 6, positive class probabilities are estimated with Equation 3. We now give formulas for the estimates of $Pr(w_i)$ and $Pr(1)$.

**Estimate of $Pr(w_i)$.** Assuming that the set of unlabeled documents is generated according to the underlying generative model, probability $Pr(w_i)$ is estimated on the set of unlabeled documents by:

$$\hat{Pr}(w_i) = \frac{N(w_i, UD)}{N(UD)} \quad (7)$$

**Estimate of $Pr(1)$.** We will consider two different estimates for $Pr(1)$. First, under the assumption that the lengths of documents are independent of the class, positive and negative documents have the same average length and $\hat{Pr}(1)$ could be set to $\hat{P}(1)$.

Second, we have seen that, given a set $D = PD \cup ND$ of labeled documents, an estimate of $Pr(1)$ is $\frac{N(PD)}{N(D)}$. We can deduce the following

equation:

$$\hat{Pr}(1) = \frac{N(PD)}{Card(PD)} \times \frac{Card(PD)}{Card(D)} \times \frac{Card(D)}{N(D)}$$

In the case where an estimate of $P(1)$ and a set $PD$ of positive documents together with a set $UD$ of unlabeled documents are given to the learner, the first term $\frac{N(PD)}{Card(PD)}$ in the previous equation can be calculated with the input set $PD$; the second term corresponds to $\hat{P}(1)$ which is given as input to the learner; and, assuming that unlabeled documents are generated according to the underlying probabilistic model, the third term can be estimated over the set $UD$ of unlabeled examples. This leads to the following estimate for $Pr(1)$:

$$\hat{Pr}(1) = \frac{N(PD)}{Card(PD)} \times \hat{P}(1) \times \frac{Card(UD)}{N(UD)}$$

When the sets $PD$ and $UD$ are quite small, it may be possible that our estimate for $Pr(1)$ is greater than 1. Thus, we bound our estimate:

$$\hat{Pr}(1) = \min\left\{ \frac{N(PD)}{Card(PD)} \times \hat{P}(1) \times \frac{Card(UD)}{N(UD)} \; ; \; \frac{1+\hat{P}(1)}{2} \right\} \quad (8)$$

Equations 3, 7 and 8 provide estimates for word probabilities appearing in Equation 6.

### Smoothing Word Probabilities

Using Equation 7, estimates for negative word probabilities $\hat{Pr}(w_i|0)$ given by Equation 6 can be rewritten:

$$\frac{N(w_i, UD) - \hat{Pr}(w_i|1) \times \hat{Pr}(1) \times N(UD)}{(1 - \hat{Pr}(1)) \times N(UD)}$$

The estimates $\hat{Pr}(w_i|0)$ can be negative. Thus, we set the negative values to 0 and normalize our estimates such that they sum to 1. Let $Z$ be the normalizing factor defined by

$$Z = \sum_{w_i \in V | Pr(w_i|0) > 0} Pr(w_i|0)$$

Using the Laplace smoothing method, estimates for negative word probabilities $\hat{Pr}(w_i|0)$ are given by:

$$\frac{1 + \max\{R(w_i); 0\} \times \frac{1}{Z}}{Card(V) + (1 - \hat{Pr}(1)) \times N(UD)} \quad (9)$$

where $R(w_i)$ is set to $N(w_i, UD) - \hat{Pr}(w_i|1) \times \hat{Pr}(1) \times N(UD)$, $\hat{Pr}(w_i|1)$ is calculated according to Equation 3, and $\hat{Pr}(1)$ is either set to $\hat{P}(1)$ or is calculated according to Equation 8.

Table 2: Naive Bayes from positive and unlabeled examples (PNB)

Given an estimate $\hat{P}(1)$ of the positive class probability $P(1)$, a set $PD$ of positive documents together with a set $UD$ of unlabeled documents, the positive naive Bayes classifier classifies a document $d$ consisting of $n$ words $(w_1, \ldots, w_n)$ as a member of the class

$$\mathsf{PNB}(d) = \operatorname*{argmax}_{c \in \{0,1\}} \hat{P}(c) \prod_{i=1}^{i=n} \hat{Pr}(w_i|c) \quad (10)$$

where the class probability estimate $\hat{P}(0)$ is set to $1 - \hat{P}(1)$, the word probability estimates are calculated according to Equation 3 for the positive class and according to Equation 9 for the negative class.

## 3 Experimental results

We consider the WebKB Course dataset[1], a collection of 1051 web pages collected from computer science departments at four universities. The binary classification problem is to identify web pages that are course home pages. The class course is designed as the positive class in our setting. In the WebKB dataset, 22% of the web pages are positive. We consider the *full-text view* which consists of the words that occur on the web page. The vocabulary is the set of words in the input data sets; no stoplist is used and no stemming is

[1] available at `http://www-2.cs.cmu.edu/afs/cs/project/theo-4/text-learning/www/datasets.html`

Table 3: results for PNB on the WebKB Course dataset when varying the number of unlabeled documents

| $p$ is set to 20 | | $p$ is set to 50 | |
|---|---|---|---|
| $u$ | error | $u$ | error |
| 20 | 27.155 | 50 | 15.265 |
| 30 | 16.597 | 100 | 8.010 |
| 40 | 12.000 | 120 | 7.485 |
| 50 | 10.353 | 130 | 7.298 |
| 60 | 8.611 | 140 | 7.265 |
| 70 | 8.698 | 150 | 7.611 |
| 80 | 8.922 | 160 | 7.576 |
| 100 | 9.586 | 170 | 7.668 |
| 150 | 13.365 | 180 | 7.693 |
| 200 | 16.048 | 200 | 8.239 |

performed. We give experimental results for our algorithm PNB when varying the number of unlabeled documents and when using different estimates for $Pr(1)$. Then, we conduct experiments to compare PNB and NB while varying the number of labeled documents. In a last set of experiments, we compare error rates when giving as input different values for the positive class probability.

### 3.1 Varying the number of unlabeled documents

We use the algorithm PNB where $Pr(1)$ is estimated using Equation 8. We set the input $\hat{P}(1)$ to 0.22. We consider two values for the number $p$ of positive documents : 20 and 50. We let vary the number $u$ of unlabeled documents. For each value of $p$ and $u$, 200 experiments are conducted. Error rates are estimated on an hold-out test set and error rates are averaged over these 200 experiments.

Experimental results (see Table 3) show that the error decreases and reaches a minimal value. We note that when the number of unlabeled documents becomes too large, performance of PNB may be poor. For a given number of positive documents, the optimal value for the number of unlabeled documents is not known. In the following, we assume that estimates will be done on a set of unlabeled documents containing approximately $Card(PD)$ positive documents. Consequently, we set the number of unlabeled documents to $Card(PD)/\hat{P}(1)$ where $PD$ is the set of positive documents and $\hat{P}(1)$ the estimate of the positive class probability. Results given in Table 3 show that this choice is not optimal from an experimental point of view on the WebKB Course dataset.

### 3.2 Estimating $Pr(1)$

We compare three variants of PNB depending on how the estimate of $Pr(1)$ is calculated. PNB takes as input $\hat{P}(1) = 0.22$ together with randomly drawn sets $PD$ and $UD$ such that $Card(UD) = Card(PD)/\hat{P}(1)$. In the first variant, $Pr(1)$ is estimated using Equation 8. In the second one, $\hat{Pr}(1)$ is set to $\hat{P}(1)$, i.e. it is supposed that the knowledge of the average length of positive documents is negligible in the classification decision. In the third one, $Pr(1)$ is estimated on the whole WebKB Course dataset of 1051 web pages and we set $\hat{Pr}(1)$ to 0.28[2].

Experimental results (see Figure 1) show that a better estimate of $Pr(1)$ slightly increases the accuracy of PNB classifiers. PNB classifiers where $\hat{Pr}(1)$ is set to $\hat{P}(1)$ perform better than PNB classifiers where $\hat{Pr}(1)$ is calculated using Equation 8 when the train set is small. Indeed the variance of the estimation of $Pr(1)$ is high when only a small number of documents are available. But, when there are enough documents (20 positive documents), the accuracy of PNB classifiers where $\hat{Pr}(1)$ is calculated using Equation 8 is close to the accuracy of PNB classifiers where $Pr(1)$ is estimated on the whole WebKB Dataset.

### 3.3 A comparison between NB and PNB

For a given number $p$, we compare: NB classifiers obtained from $p$ labeled documents; PNB classifiers obtained with input $\hat{P}(1) = 0.22$, $p$

[2]Note that under the assumption that the length of documents is independent of the class, $Card(PD)/Card(D)$ and $N(PD)/N(D)$ are unbiased estimates of $Pr(1)$. On the WebKB Course dataset, we find respectively 0.22 and 0.28 which suggests that this assumption could be not correct.

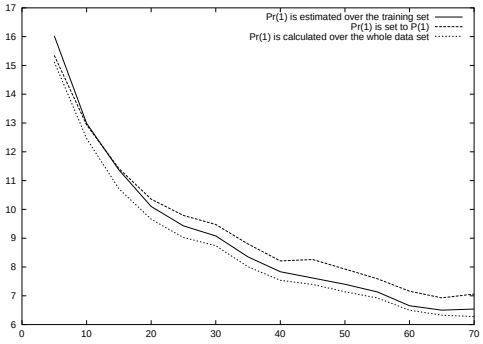



Figure 1: Comparison of PNB with three different estimates of $Pr(1)$. Error rates are averaged over 200 experiments

positive documents and $N \simeq p \times 1/0.22$ unlabeled documents; NB classifiers obtained from $N$ labeled documents. We use algorithm PNB where $\hat{Pr}(1)$ is estimated using Equation 8. For each value $p$ and each algorithm, 200 experiments are conducted. Error rates are estimated on an hold-out test set and are averaged over the 200 experiments. Error rates are given together with standard deviation.

Experimental results (see Table 4 and Figure 2) show that PNB classifiers outperform NB classifiers obtained from $p$ labeled documents. These experimental results are quite promising showing that $p$ positive examples completed with unlabeled examples have a higher value than $p$ labeled examples, at least for small values of $p$.

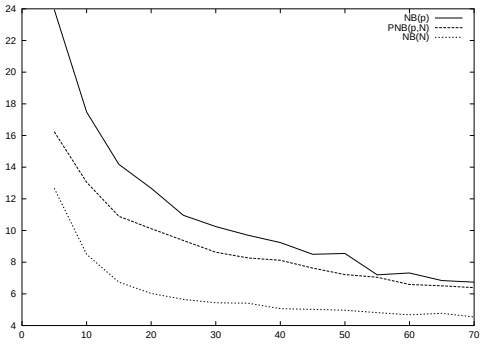



Figure 2: Comparison of $\mathsf{NB}_p$, $\mathsf{PNB}_{p,N}$ and $\mathsf{NB}_N$.

Table 4: A comparison between NB and PNB.

| $p$ | $N$ | $\mathsf{NB}_p$ | $\mathsf{PNB}_{p,N}$ | $\mathsf{NB}_N$ |
|---|---|---|---|---|
| 5 | 22 | 23.95(12.4) | 16.24(12.67) | 12.67(4.72) |
| 10 | 45 | 17.49(7.00) | 13.05(4.68) | 8.50(3.56) |
| 15 | 68 | 14.18(5.55) | 10.90(4.13) | 6.74(2.40) |
| 20 | 91 | 12.67(4.72) | 10.12(3.70) | 6.03(1.95) |
| 25 | 114 | 10.96(4.26) | 9.37(3.39) | 5.65(1.79) |
| 30 | 137 | 10.25(4.51) | 8.63(2.95) | 5.44(1.64) |
| 35 | 159 | 9.70(4.26) | 8.27(2.74) | 5.41(1.58) |
| 40 | 182 | 9.24(4.22) | 8.12(2.61) | 5.07(1.45) |
| 45 | 205 | 8.50(3.56) | 7.63(2.52) | 5.02(1.49) |
| 50 | 228 | 8.55(3.73) | 7.22(2.39) | 4.97(1.38) |
| 55 | 251 | 7.20(2.97) | 7.05(2.12) | 4.81(1.42) |
| 60 | 274 | 7.32(3.18) | 6.59(1.83) | 4.68(1.35) |
| 65 | 297 | 6.84(2.45) | 6.51(1.94) | 4.77(1.37) |
| 70 | 319 | 6.74(2.40) | 6.39(1.95) | 4.54(1.29) |

### 3.4 Giving an estimate of the positive class probability

We use the algorithm PNB where $Pr(1)$ is estimated using Equation 8. We consider two values for the number $p$ of positive documents : 20 and 50. An estimate of the positive class probability on the whole WebKB Dataset is $\hat{P}(1) = 0.22$. We let vary the estimate for the positive class probability. PNB takes as input $\hat{P}(1)$ together with randomly drawn sets $PD$ and $UD$ such that $Card(UD) = Card(PD)/\hat{P}(1)$. $\hat{P}(1)$ takes value from 0.12 to 0.38 by step 0.02. For each value of $\hat{P}(1)$, 200 experiments are conducted. Error rates are estimated on an hold-out test set and error rates are averaged over these 200 experiments.

Experimental results are given in Table 5. They show that sufficiently accurate classifiers are obtained with rough estimates of $P(1)$. For instance, an estimate of $P(1)$ could be chosen between 0.2 and 0.3.

## 4 Conclusion

We have shown that text classification from positive and unlabeled data is feasible and that positive documents and labeled documents may have a comparable value as soon as the former are completed with enough un-

Table 5: PNB classifiers with different input values for $\hat{P}(1)$.

| | $p$ is set to 20 | $p$ is set to 50 |
|---|---|---|
| $\hat{P}(1)$ | error | error |
| 0.12 | 16.74 | 13.47 |
| 0.14 | 15.12 | 11.37 |
| 0.16 | 13.77 | 9.99 |
| 0.18 | 11.93 | 8.88 |
| 0.20 | 10.76 | 8.00 |
| 0.22 | 10.60 | 7.22 |
| 0.24 | 9.66 | 7.18 |
| 0.26 | 9.25 | 6.71 |
| 0.28 | 9.96 | 6.78 |
| 0.30 | 10.21 | 7.23 |
| 0.32 | 11.29 | 8.18 |
| 0.34 | 12.41 | 9.22 |
| 0.36 | 12.69 | 9.70 |
| 0.38 | 13.74 | 11.26 |

labeled documents. As in the semi-supervised framework, unlabeled data are supposed to be freely available, the experimental results are promising but we need to apply our algorithms to other data sets. Following [7], it would be interesting to design algorithms from positive and unlabeled documents when the positive class probability is not given as input to the learner. Also, we intend to adapt the co-training setting from Blum and Mitchell [1] to the framework of learning from positive and unlabeled documents.

## Acknowledgements

This research was partially supported by: "CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: axe TACT, projet TIC"; fonds européens FEDER "TIC - Fouille Intelligente de données - Traitement Intelligent des Connaissances" OBJ 2-phasing out - 2001/3 - 4.1 - n 3; "projet DATADIAB - ACI télémédecine et technologies pour la santé".

## References

[1] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. 11th Annu. Conf. on Comput. Learning Theory*, pages 92 – 100, 1998.

[2] F. DeComité, F. Denis, R. Gilleron, and F. Letouzey. Positive and unlabeled examples help learning. In *Proc. 10th International Conference on Algorithmic Learning Theory*, pages 219 – 230, 1999.

[3] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier using zero-one loss. *Machine Learning*, 29:103 – 130, 1997.

[4] T. Hofmann. Text categorization with labeled and unlabeled data: A generative model approach. In *Working Notes for NIPS 99 Workshop on Using Unlabeled Data for Supervised Learning*, 1999.

[5] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proc. 16th International Conference on Machine Learning*, pages 200 – 209, 1999.

[6] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proc. 25th ACM Symposium on the Theory of Computing*, pages 392 – 401, 1993.

[7] F. Letouzey, F. Denis, and R. Gilleron. Learning from positive and unlabeled examples. In *Proc. 11th International Conference on Algorithmic Learning Theory*, pages 71 – 85, 2000.

[8] D. D. Lewis. Naive (bayes) at forty: the independence assumption in information retrieval. In *Proc. 10th European Conference on Machine Learning*, pages 4 – 15, 1998.

[9] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Proc. 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 81 – 93, 1994.

[10] Kamal Nigam and Rayid Ghani. Analyzing the applicability and effectiveness of co-training. In *Proc. 9th International Conference on Information and Knowledge Management*, pages 86 – 93, 2000.

[11] Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom M. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103 – 134, 2000.

**Résumé**

Le programme de recherches présenté dans cette synthèse s'inscrit dans la double problématique de l'étude des langages d'arbres et de l'apprentissage automatique à partir de données arborescentes.

À la base de ce travail se trouve la question de l'accès et de la manipulation automatique d'informations au format XML au sein d'un réseau d'applications réparties dans Internet. La réalisation de ces applications est toujours du ressort de programmeurs spécialistes d'XML et reste hors de portée de l'utilisateur final. De plus, les développements récents d'Internet poursuivent l'objectif d'automatiser les communications entre applications s'échangeant des flux de données XML. Le recours à des techniques d'apprentissage automatique est une réponse possible à cette situation.

Nous considèrons que les informations sont décrites dans un langage XML, et dans la perspective de ce mémoire, embarquées dans des données structurées sous forme arborescente. Les applications sont basées alors sur des opérations élémentaires que sont l'interrogation ou les requêtes dans ces documents arborescents ou encore la transformation de tels documents.

Nous abordons alors la question sous l'angle de la réalisation automatique de programmes d'annotation d'arbres, permettant de dériver des procédures de transformation ou d'exécution de requêtes. Le mémoire décrit les contributions apportées pour la manipulation et l'apprentissage d'ensembles d'arbres d'arité non bornée (comme le sont les arbres XML), et l'annotation par des méthodes de classification supervisée ou d'inférence statistique.

**Mots clés**

Langages d'arbres, automates d'arbres, apprentissage automatique, XML, champs aléatoires, classification supervisée, extraction d'information.